UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

MARCAÇÃO SEMÂNTICA DE PÁGINAS WEB APOIADA POR PARSERS DE DEPENDÊNCIAS GRAMATICAIS

Rúben Alberto Mendes Simões dos Reis

MESTRADO EM ENGENHARIA INFORMÁTICA
Especialização em Sistemas de Informação

2010

UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

MARCAÇÃO SEMÂNTICA DE PÁGINAS WEB APOIADA POR PARSERS DE DEPENDÊNCIAS GRAMATICAIS

Rúben Alberto Mendes Simões dos Reis

DISSERTAÇÃO

Projecto orientado pelo Prof. Doutor António Horta Branco

MESTRADO EM ENGENHARIA INFORMÁTICA
Especialização em Sistemas de Informação

2010

Agradecimentos

Gostaria de agradecer aos meus pais Luís Reis e Ana Simões, aos meus avós maternos Luís Lopes e Lícia Mendes, à minha namorada Mónica Magalhães, ao meu cachorro Nilo, aos meus amigos António, Freire, David, Serra, Sérgio, Fábio, Marcos, Ricardo, Coelho, à malta das quintas-feiras de Futebol e claro, aos elementos do Grupo NLX João Silva, Roda Del Gáudio, Francisco Costa, Sara Silveira, Sílvia Pereira, Marcus Guelpeli, Clara Pinto, Catarina Carvalheiro, Sérgio Castro e Patrícia Gonçalves por todo o apoio e ajuda que me deram ao longo deste trabalho.

Gostaria também de agradecer ao meu professor orientador António Horta Branco por toda a paciência e ajuda, e à Fundação da Ciência e Tecnologia pelo financiamento deste trabalho.

O meu sincero Obrigado a todos

Resumo

Com o crescimento exponencial da informação na Web, torna-se necessário cada vez mais que o acesso à informação não só seja rápido, como eficiente. A procura por informação através da ocorrência de palavras-chave é o método usado pelos motores de busca na web mais conhecidos. Contudo a busca por informação na Web pode ser optimizada usando uma representação semântica da informação pela qual se procura.

Este trabalho apresenta o desenvolvimento de uma ferramenta para a anotação semântica de páginas Web escritas em Português, apoiada por Analisadores de Dependências Gramaticais. Essa ferramenta recebeu o nome de *Marcador Semântico* e tem a capacidade de atribuir uma representação semântica a frases inseridas num texto e deixar essa representação semântica registada na linguagem de marcação RDF/XML.

Neste trabalho, também é documentado uma ferramenta Web, adicionada ao repositório de ferramentas on-line do grupo NLX, da Faculdade de Ciências Universidade de Lisboa. Esta ferramenta, chamada de *LX Dep Parser*, é uma Analisador de Dependências Gramaticais e tem a finalidade de devolver ao utilizador uma representação das dependências gramaticais entre as palavras da frase.

Palavras-chave: Semântica, Parser, Dependências, RDF/XML, NLX

Abstract

With the exponential growth of the information in the Web, it becomes increasingly necessary that access to information be not only fast, but efficient. The search for information by means of the occurrences of keywords is the method used by Web search engines. However, the search for information on the Web can be optimized using a semantic representation of the information that is being sought.

The present work presents a tool for semantic annotation of Web pages written in Portuguese, supported by Dependency Parsers. This tool, named *Marcador Semântico*, has the ability to provide a semantic representation for a number of sentences ocurring in a text, and encode the semantic representation of these sentences in the markup language RDF/XML. This work also presents a web tool, added to the repository of online tools of the NLX group, the Faculty of Sciences, University of Lisbon. This tool, called LX Dep Parser, is a Grammatical Dependency Parser and aims at returning to the user a representation of grammatical dependencies among the words of the input sentence.

Keywords: Semantic, Parser, Dependency, RDF/XML, NLX

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Motivação

As exigências na procura de informação tornam-se cada vez maiores num mundo em que essa informação está em constante crescimento. A representação semântica da informação presente na Web é possível e torna-se cada vez mais necessário pois com o rápido crescimento dessa informação, passa a existir uma necessidade do ser humano em ter um acesso eficiente à informação que procura.

Na área do Processamento da Linguagem Natural, para que seja possível trabalhar com a semântica de um texto, é necessário uma representação semântica desse mesmo texto de modo a que seja possível efectuar algum tipo de processamento sobre esse tipo de informação.

Nos últimos anos têm merecido um acrescido interesse, modelos de parsing dos quais resultem estruturas de informação que representem a semântica de textos escritos numa linguagem natural. Este interesse é facilmente justificado dado que a criação destas estruturas de informação que contêm a representação semântica de um dado texto apresenta uma boa eficiência em termos de complexidade computacional e também em termos de taxa de acerto.

Assim sendo, a estrutura que contém a representação semântica de um dado texto será a chave que tornará possível melhorar os dispositivos de busca de informação na Web.

1.2 Contribuição

Esta dissertação documenta um projecto de Engenharia Informática do Mestrado em Engenharia Informática do Departamento de Informática da FCUL,[1], realizou uma ferramenta capaz de obter uma estrutura de informação semântica a partir de um dado texto na Língua Portuguesa, e efectuar a extracção da representação semântica a partir da estrutura de informação semântica desse mesmo texto. Essa ferramenta foi denominada de

[1]Faculdade de Ciências Universidade de Lisboa

Marcador Semântico.

Para ser possível gerar uma representação semântica de um dado texto, utilizei um tipo de ferramenta chamada de analisador (parser) de dependências. A estrutura de informação criada pelo parser de dependências para cada frase do texto é também conhecida por grafo de dependências, onde cada palavra se relaciona com outra palavra através de uma função gramatical ou semântica.
Esse tipo de relações gramaticais e/ou semânticas entre as palavras de uma frase, serão registados na linguagem RDF/XML, com a finalidade de motores de busca ou agentes artificiais poderem analisar, comparar e processar este tipo de informação. Uma vez que interessa registar a relação semântica entre duas palavras, a linguagem RDF/XML será utilizada, pois sendo que esta linguagem é orientada a triplos (triplos esses que são constituidos por: Sujeito, Predicado, Objecto) iremos querer registar duas palavras (que correspondem ao Sujeito e Objecto do triplo), com uma relação "Predicado".

A ferramenta *Marcador Semântico*, capaz de registar a representação semântica de um texto, constitui uma inovação dentro da área que é o Processamento da Linguagem Natural uma vez que este tipo de ferramentas é inexistente.

É importante referir que o contributo para o repositório de ferramentas Web do Grupo NLX, foi efectuado com mais uma ferramenta chamada de *LX Dep Parser*, que possui o objectivo de devolver um grafo de dependências para uma dada frase inserida por um utilizador.

1.3 Estrutura do documento

Este documento está estruturado da seguinte forma:

- **Capítulo 1** - Para além da "Motivação" e "Contribuição", neste capítulo será abordado o conceito de parser de dependências bem como um dos formatos (CoNLL) que este tipo de ferramentas utiliza, assim como o conceito de projectividade que pode surgir ou não, nos grafos de dependências.
- **Capítulo 2** - Neste capítulo, serão descritos os parsers de dependências disponíveis e qual o parser escolhido para concretizar a ferramenta *Marcador Semântico* e *LX Dep Parser*.
- **Capítulo 3** - Neste capítulo será descrita a ferramenta *LX Dep Parser*.
- **Capítulo 4** - Será explicada neste capítulo, a ferramenta *Marcador Semântico*.

1.4 Parsing de Dependências

Um parser de dependências é uma ferramenta que recebe como input um ficheiro de texto escrito numa linguagem natural, e devolve um grafo de dependências que contém as

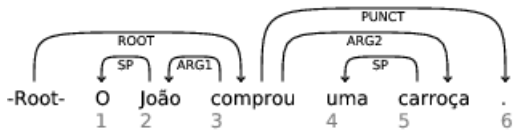


Figura 1.1: Anotação dos relações semânticas das palavras na frase: "O João comprou uma carroça."

funções gramaticais entre as palavras das frases desse texto.

Através de um modelo produzido pelo treino, com um corpus de frases e suas estruturas de dependências, o parser de dependências procede à tarefa de parsing sobre o texto dado como input. O resultado básico consiste em calcular arcos de dependências entre pares de palavras e respectivas etiquetas gramáticas. Os arcos de dependências e respectivas etiquetas gramáticas e/ou semânticas constituirão o grafo de dependências gramaticais da frase em questão.

Segundo [8], as gramáticas e teorias de gramáticas podem ser classificadas de acordo com a unidade básica da estrutura da frase que consideram: o sintagma ("Phrase Structure Grammar") ou a dependência gramatical entre duas palavras ("Dependency Grammar"). Os parsers de Gramáticas Sintagmáticas devolvem uma estrutura baseada nos constituintes da respectiva frase de entrada, enquanto que um parser de dependências devolve as funções gramaticais e/ou semânticas entre palavras dessa frase, na forma de um grafo de dependências:

Na Figura 1.1, pode ser observado o resultado de uma frase analisada através de um parser de dependências, em que as relações entre as palavras (arcos de dependências) são:

- "O" é especificador de *João* (Specifier - SP).
- "João" é o primeiro argumento do verbo *comprou* (Argument 1 - ARG1).
- "comprou" é o núcleo da frase (Root - ROOT).
- "uma" é o especificador de *carroça* (Specifier - SP).
- "carroça" é o segundo argumento do verbo *comprou* (Argument 2 - ARG2).
- "." é a pontuação final da frase (Punctuation - PUNCT).

Este grafo de dependências pode ser visto como uma árvore de dependências em que o núcleo (palavra de onde sai o arco de dependência) possui um ou mais filhos (palavras para onde o arco de dependência aponta).

Este tipo de grafos, para representarem uma estrutura de dependências deverão sempre ser bem formados. Um grafo de dependências é bem formado se e só se:

- Um e só um nó for a raiz (para a frase em questão).

- Se o grafo for ligado, ou seja, se a partir do núcleo da frase é possível chegar a todos os restantes nós (palavras).
- Todo o nó, excepto a raiz, tiver um nó "pai".
- O grafo for acíclico.

Os parsers de dependências recebem frases de texto como input. A estas frases, deverá estar agregada informação morfo-sintáctica para auxiliar o parser de dependências nas suas decisões de parsing, de modo a constituir o grafo de dependências de uma dada frase.
Na próxima secção explicar-se-á um formato generalizado para codificar estruturas de dependências gramaticais, usado por este tipo de ferramentas.

1.5 Formato CoNLL

Os parsers de dependências gramaticais efectuam a tarefa de parsing que consiste em calcular os grafos de dependências de cada frase fornecida no input, podendo esse mesmo input estar em vários formatos. No entanto existe um formato em particular que foi generalizado para este tipo de ferramentas, chamado de CoNLL. *CoNLL* (Conference on Natural Language Learning)[2] é uma conferência que se realiza anualmente onde se abordam vários tópicos relacionados com a aprendizagem automática aplicada ao Processamento em Linguagem Natural.

Esta conferência tem vindo a decorrer ao longo dos anos, desde 1997, tendo a última conferência ocorrido em Fevereiro de 2009. A Conferência de 2010 está marcada para dias 15 e 16 de Julho, a decorrer em Uppsala, na Suécia.

Esta conferência está classificada como a décima-sétima mais importante,[3] na área da Inteligência Artificial. Com o decorrer das várias edições desta conferência, um dos temas que foi sendo definido foi a concepção de analisadores de dependências para que fosse possível a análise de dependências gramaticais e respectiva etiquetação de uma frase.

Como tal, foi definido um formato de input para os analisadores de dependências gramaticais desta conferência, em que o ficheiro de input contém: frases separadas por uma linha em branco, cada palavra ou símbolo (por exemplo pontuação) da frase numa linha sendo que cada linha contém dez colunas, separadas por uma espaçamento tabular (*Tab*). O formato CoNLL 2006 é explicado com o auxílio da Tabela 1.1.
Em termos práticos, tomando em conta a seguinte frase:

A Maria tem razão.

[2] `http://ifarm.nl/signll/conll/`
[3] `http://www.cs-conference-ranking.org/conferencerankings/topicsii.html`

Número do Campo:	Nome de Campo:	Descrição:
1	ID	Contador de palavras, de acordo com a sua ordem (da esquerda para a direita) de ocorrência na frase. Começa com o valor 1 para cada nova frase.
2	Form	Forma da palavra ou símbolo de pontuação.
3	Lemma	Lema da palavra, ou uma sublinha se não estiver disponível.
4	CPOSTAG	Etiqueta morfo-sintáctica de alta granularidade, em que o conjunto de etiquetas depende da linguagem natural em questão.
5	POSTAG	Etiqueta morfo-sintáctica de baixa granularidade, em que o conjunto de etiquetas depende da linguagem natural em questão. Se não estiver disponível, é dada a etiqueta morfo–sintáctica de alta granularidade.
6	FEATS	Conjunto não organizado de características sintácticas e/ou morfológicas (dependendo da linguagem natural) separadas por uma barra vertical (—), ou uma sublinha se não estiver disponível.
7	HEAD	Head (núcleo) do símbolo corrente, que é identificado por um inteiro. De referir que dependendo da anotação original do treebank, poderá haver vários símbolos com um ID igual a zero.
8	DEPREL	Relação de dependência com a Head. O conjunto de dependências depende da linguagem natural em questão. De referir que dependendo da anotação original do treebank, a relação de dependência pode ser uma etiqueta atribuída, proveniente da anotação original do treebank, ou ser simplesmente 'ROOT'.
9	PHEAD	Head (núcleo) projectiva do símbolo corrente, que ou é representado por um inteiro (zero incluído), ou por uma sublinha, se estiver indisponível. De notar que ao depender da anotação original do treebank, poderá existir vários símbolos com um ID de 0. A estrutura de dependência resultante da coluna PHEAD, é projectiva (embora tal característica não esteja disponível em todas as línguas).
10	PDEPREL	A relação de dependência da PHEAD, ou uma sublinha se não estiver disponível. O conjunto de relações de dependências depende, da língua em questão. De notar que se depender da anotação do treebank original, a relação de dependência pode conter algum significado, ou então ser representado por 'ROOT'

Tabela 1.1: Formato CoNLL 2006

1	A	_	DA	DA	fs	2	SP	_	_
2	Maria	_	PNM	PNM	_	3	ARG1	_	_
3	tem	TER	V	V	_	0	ROOT	_	_
4	razão	RAZÃO	CN	CN	gs	3	ARG2	_	_
5	.	_	PNT	PNT	_	3	PUNCT	_	_

Tabela 1.2: Grafo de dependências em formato CoNLL 2006, da frase: "A Maria tem razão."

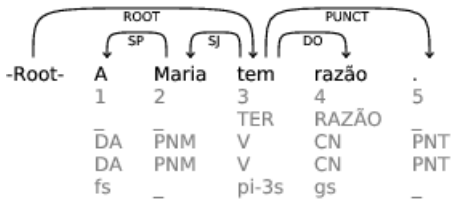


Figura 1.2: Ilustração do grafo de dependências da frase: "A Maria tem razão."

No formato CoNLL 2006, o grafo de dependências desta frase (com os arcos de dependência e respectivas etiquetas gramaticais atribuídas) ficaria como está mostrado na Tabela 1.2. A descrição do conteúdo de cada coluna encontra-se na Tabela 1.1.

Em que a ilustração do grafo de dependências com base na frase acima em formato CoNLL 2006 ficaria de acordo com a Figura 1.2.

Uma vez que só são utilizadas etiquetas morfo-sintácticas de granularidade fina no corpus de dependências do NLX que usámos para este trabalho, é colocada a mesma etiqueta morfo-sintáctica de uma palavra nos campos quatro e cinco.

Será este o formato do corpus de dependências com etiquetas gramaticais e semânticas que será usado para treinar o parser de dependências, por forma a que o parser devolva um grafo de dependências, para que seja possível proceder a uma extracção da representação semântica de uma frase (consequentemente de um texto) e para que posteriormente se registe essa mesma representação semântica do texto, em linguagem RDF/XML.

É importante referir que as colunas ID, FORM, CPOSTAG, POSTAG, HEAD e DEREL devem conter valores significativos em vez de, por exemplo, sub-linha, aquando do treino dos parsers de dependências gramaticais.
Também importa referir que os últimos dois campos, se referem à projectividade de uma frase. A finalidade destas colunas seria a de serem preenchidas pelo parser de dependências em questão, na tarefa de parsing, caso a frase a ser analisada fosse projectiva. Contudo, não foi verificado o preenchimento destas duas colunas após a tarefa de parsing, utilizando os parsers de dependências gramaticais analisados no Capítulo dois. Uma vez que estas duas colunas não apresentam quaisquer tipo de contributo para a tarefa de parsing dos parsers de dependências gramaticais, os corpus de treino em formato CoNLL 2006 terão estas duas colunas preenchidas com uma sub-linha.

As duas colunas ”PHEAD“ e ”PDEPREL“, referem-se à projectividade de uma frase, como anteriormente referido. Como tal, passaremos então a explicar o conceito de ”projectividade“ na próxima secção.

1.6 Projectividade

Um grafo de dependências de uma frase é uma representação linguística adoptada por um grande número de teorias de gramática e formalismos linguísticos, que partilham um número de pressupostos sobre estruturas sintácticas. Um grafo de dependências é constituído por nós lexicais, em que cada nó é dependente de um outro nó (excepto se for a raiz). Um grafo/árvore de dependências deve ser segundo [14]:

- Acíclico
- Conectado
- Projectivo

Existem várias definições de projectividade, mas uma vez que todas elas são equivalentes, passei a utilizar a definição de [14], definindo a projectividade em termos de adjacência (ou seja em termos da posição relativa entre as palavras, de uma dada frase).

- Um grafo de dependências é projectivo se e só se cada nó dependente (palavra) for adjacente ao seu núcleo (Head).
- Dois nós (palavras) n e n' são adjacentes no grafo se e só se todo o nó n'' que ocorre no grafo entre n e n', for “dominado” por n ou por n'. Este domínio entende-se por exemplo, como o caso em que um nó m ocorra entre n e n', mas que o seu núcleo seja n''. Se n'' tiver como núcleo n ou n', então considera-se que m é dominado por n ou n', o que significa que existe projectividade no grafo de dependências em questão.

Exemplos de frases não projectivas podem ser encontradas facilmente na língua alemã. Na Figura 1.3 podemos observar um exemplo de uma frase “não projectiva”, escrita na língua alemã.
Analisando o arco de dependência na Figura 1.3, que envolve os nós 1 (“Fur”) e 8 (“Nachweis”), podemos observar que o nó 6 (“bisher”) que se encontra dentro do arco de dependência formado entre o nós 1 e 8. No entanto, o nó 6 depende do nó 9 (“geliefert”). Nó este (9 - “geliefert”) que está fora do arco de dependência formado pelos nós 1 e 8.
O que de acordo com a definição de projectividade escrita anteriormente, torna esta frase não projectiva (que em caso geral se caracteriza por existir nos grafos de dependências, cruzamento entre os arcos). Para finalizar, realizei uma experiência para verificar a eficiência

Figura 1.3: Anotação retirada do corpos TuBa-D/Z treebank. Tradução: "Para esta alegação, Beckmeyer não forneceu nenhuma prova."

de algoritmos de parsing projectivos ([14], [19]) sobre grafos de dependências projectivos. Para tal, foram excluídas as frases não projectivas do corpus de dependências do NLX (cento e quatorze frases não projectivas), e utilizou-se um algoritmo projectivo de parsing e um algoritmo não projectivo de parsing:

- Com o parser de dependências MST, foi verificado que (correndo o parser com as definições por omissão) num teste "Ten Fold Cross Validation", com o algoritmo de parsing não projectivo foi obtido o valor de 0.8909 para a métrica LAS, decorridos 3106 segundos de treino e parsing da ferramenta, enquanto que com o algoritmo projectivo foi obtido o valor de 0.8922 para a métrica LAS, decorridos 2769 segundos de treino e parsing da ferramenta.

- Com o parser de dependências Malt, foi verificado que (correndo o parser com as definições por omissão) num teste "Ten Fold Cross Validation" com o algoritmo não projectivo "Convigton Non Projective" foi obtido o valor de 0.8871 para a métrica LAS, decorridos 533 segundos de treino e parsing da ferramenta, enquanto que com o algoritmo projectivo "Nivre - arc standard" foi obtido o valor de 0.8902 para a métrica LAS, decorridos 238 segundos de treino e parsing da ferramenta.

(Nota: o tipo de teste "Ten Fold Cross Validation", a métrica de avaliação LAS, MST Parser e Malt Parser serão conceitos devidamente explicados, no Capítulo 2.)

1.7 Relações Semânticas

Como referido, para que seja possível construir o *Marcador Semântico* a fim de se extrair a representação semântica de um dado texto (e posteriormente registar a semântica das páginas web escritas na Língua Portuguesa em linguagem RDF/XML) será necessário um parser de dependências através do qual se obtém o grafo de dependências, para cada frase de um dado texto.
Este grafo de dependências terá de ter definidos os arcos de dependências, bem como as respectivas etiquetas gramaticais e semânticas, entre as palavras de uma frase. Para que os grafos de dependências (parse final de uma dada frase) contenham principalmente as relações semânticas entre as palavras, é necessário que o corpus usado para treinar o

parser de dependências contenha os arcos de dependências definidos com as respectivas etiquetas semânticas.

A diferença entre as etiquetas gramaticais e as etiquetas semânticas está no tipo de relação que se estabelece entre as palavras dentro de uma frase, sendo que as etiquetas gramaticais expressam relações gramaticais entre os constituinte de uma frase.

O termo "relação gramatical" é um conceito que tem definições um pouco variadas, mas bastante semelhantes.
Para um futuro esclarecimento, segundo [10], uma oração (como por exemplo, uma frase simples: "O João comprou uma carroça.") contém dois termos importantes: o *Predicado* ("comprou uma carroça."), que é uma sequência de constituintes formado por um predicador e respectivo(s) argumento(s), ou constituinte(s), interno(s); e o *Sujeito* ("O João") que se relaciona directamente com o *Predicado*.

Segundo [1], o termo relação gramatical refere-se às propriedades morfo-sintácticas que relacionam um argumento com uma oração, como por exemplo, o Sujeito ou Objecto Directo. Termos alternativos como "função sintáctica", ou "papel sintáctico" podem ser usados evidenciando o facto que as relações gramaticais são definidas pela forma como argumentos (constituintes) estão integrados sintacticamente numa oração, por exemplo, funcionando como Sujeito, Objecto Directo, etc. Seja qual for a terminologia, o conceito de "relação gramatical" mantém-se.

Uma definição semelhante é apresentada por [6], que defende que os constituintes de uma combinação de palavras desempenham certas funções sintácticas na frase a que pertencem.
Assim, numa oração (ou frase simples), o SV (Sintagma Verbal - a expressão que tem como constituinte central o verbo e que denota uma propriedade ou relação, dinâmica ou não dinâmica) tem a função sintáctica de predicado e o SN ou a F constituinte imediato da frase (Sintagma Nominal ou Frase - a expressão nominal/frásica a que é atribuído tal predicado), tem a relação gramatical de sujeito.

Outra definição de "relação gramatical", segundo [10], considerando as seguintes frases:

- (a) O jornalista contou as novidades aos amigos.

- (b) A novidade aos amigos o jornalista contou.

(1a) é uma frase básica do português que pode caracterizar-se sintacticamente, numa primeira abordagem, como uma sequência em que:

- cada constituinte tem uma dada relação gramatical;

- os constituintes ocorrem segundo uma dada ordem linear.

Esta caracterização sintáctica reporta-se à forma final das frases. Em línguas como o português, a relação gramatical dos constituintes é o principal factor que determina a ordem da sua ocorrência.

Passemos então a descrever as relações gramaticais, segundo [10], referidas anteriormente, bem como outras relações gramaticais: *Objecto Directo* e *Complemento Oblíquo*, para um melhor entendimento das relações gramaticais entre constituintes, numa dada frase:

- **Sujeito** - Trata-se da relação gramatical central a que é dada maior proeminência sintáctica, em frases básicas como a frase na Figura 1.4.
  Tipicamente realiza-se fora do predicado, sendo, no entanto, determinado pelo verbo.
  Deste modo, o constituinte da frase na Figura 1.4) "O João" é o sujeito gramatical da frase.

- **Predicado** - De um modo geral, uma oração consiste numa frase simples em que (considerando a frase da Figura 1.4) o predicado contém pelo menos um elemento verbal. Assim na frase: "O João comprou uma carroça" o *Predicado* "comprou uma carroça." é constituído por um verbo ("comprou") e por um argumento interno ("uma carroça").
  Um predicado para além de ser constituído por um verbo, pode também ser constituído por:

  - **Objecto Directo** - Têm esta relação gramatical os argumentos internos directos de verbos (de dois ou três lugares como, por exemplo, os verbos "dar", "oferecer"). Como tal, podemos observar na frase da Figura 1.4, a relação de *Objecto Directo* que envolve "uma carroça" é atribuída pelo verbo "comprou".
  - **Objecto Indirecto** - O constituinte que tem esta relação gramatical é tipicamente argumento interno do verbo (de dois ou três lugares como, por exemplo, os verbos "dar", "oferecer"). Considerando a seguinte frase: "O João ofereceu um CD ao Pedro.", podemos ver que o constituinte "ao Pedro" é o *Objecto Indirecto* da frase.
  - **Complemento Oblíquo** - As relações gramaticais que se estabelecem com o verbo através do auxílio de preposições (por exemplo, "na", "com", "em"), são chamadas de "oblíquas". Veja-se por exemplo a frase: "O João pôs o livro na estante". O *Complemento Oblíquo* da frase requisitado pelo verbo é "na estante".
  - **Modificador** - Um modificador é um constituinte cuja presença na frase não é obrigatória, ou seja, não é exigido por nenhum outro constituinte, mas que estando presente na frase pode modificar verbos, nomes (por exemplo), ou até

Etiqueta gramatical	**Significado**
C	Complemento
DO	Objecto Directo
IO	Objecto Indirecto
M	Modificador
N	Relação de palavras de nome próprio
OBL	Complemento Oblíquo
PRD	Predicador
SJ	Sujeito
SP	Especificador
COORD	Coordenação
PUNCT	Pontuação

Tabela 1.3: Tabela com as etiquetas gramaticais presentes no corpus de dependências do Grupo NLX.

Figura 1.4: Anotação das relações gramaticais das palavras na frase: "O João comprou uma carroça."

mesmo, toda uma frase. Considerando a seguinte frase: "Os Polícias trabalham, sem fardas.", o modificador do verbo trabalham é "sem fardas".

No entanto, considerando a Tabela 1.3, as etiquetas gramaticais adoptadas ilustram informação gramatical a dois níveis: a um nível básico, que traduz relações entre as palavras dentro de um constituinte (SP, C, N, PRD, COORD, PUNCT), e a um nível superior, que apresenta as relações entre constituintes, (SJ, OBL, IO, DO).

Com algumas das relações gramaticais descritas anteriormente e com base na frase na Figura 1.4 anotada apenas com as etiquetas gramáticas, é possível observar as relações gramaticais entre as palavras da frase:

- "O" - Especificador de *João* (Specifier - SP).
- "João" - Sujeito do verbo *comprou* (Subject - SJ).
- "comprou" - Núcleo da frase (Root - ROOT).
- "uma" - Especificador de *carroça* (Specifier - SP).
- "carroça" - Objecto directo do verbo *comprou* (Direct Object - DO).
- "." - Pontuação final da frase (Punctuation - PUNCT).

Já etiquetas semânticas definem relações semânticas que permitem caracterizar o tipo de relação semântica decorrente do relacionamento entre os constituintes de uma frase.

O termo "relação semântica", segundo [11], refere que as relações gramaticais como Sujeito e Objecto Directo nem sempre correspondem de uma maneira natural às relações semânticas existes entre um verbo e os sintagmas nominais (seus argumentos), isto é, entre um verbo e os sintagmas nominais por ele seleccionados.
Os argumentos de um verbo têm uma determinada interpretação semântica (ou papel temático) relacionada com a própria interpretação do verbo que os selecciona. Às relações semânticas é também atribuído o nome de "papéis temáticos" pois estas relações pressupõem a existência de uma relação semântica central - o Tema - sendo que, toda a frase possui um Tema. [4]

A anotação do corpus de dependências do Grupo NLX com etiquetas semânticas (Tabela 1.4), teve por base [12], em que as relações semânticas definidas são as seguintes:

- **Argumento** - Aplica-se a palavras que são argumento de um nome, adjectivo ou verbo. X representa o número do argumento que uma palavra constitui nome, adjectivo ou verbo.
  Por exemplo: "O **João** partiu."

- **Agente causativo de verbos com alternância causativa** - Aplica-se ao causador da acção do verbo.
  Por exemplo: "O **pirata** afundou o barco."

- **Modificador de Localização** - Aplica-se a localizações espaciais, quer físicas, quer abstractas.
  Por exemplo: "O Pedro mora na **av. da Liberdade**", "O Pedro referiu-se ao incidente, **no seu discurso**"

- **Modificador de Extensão** - Aplica-se a strings que determinem uma extensão, sobretudo numérica. Engloba medidas, percentagens, quantificadores e termos comparativos.
  Põe exemplo: "O desemprego subiu **15%**"; "A melancia pesava **2kg**"; "O atleta correu **2 metros**"; "A Maria engordou **bastante** no último mês"; "o Pedro gastou **mais do que o previsto**"

- **Modificador de Advérbio** - Engloba todas as strings que não se possa incluir nas restantes etiquetas.
  Por exemplo, genitivos como "a casa da Maria"

- **Modificador de Causa** - Indica a causa/razão da acção.
  Por exemplo: "A Maria chumbou porque **errou todas as perguntas do teste**"

[4] Para uma cobertura mais detalhada deste assunto ver o Capítulo 5 de [11]

- **Modificador Temporal** - Localiza a acção na linha do tempo e engloba *Frequência*, *Duração* e *Repetição*.
  Por exemplo: "O crime aconteceu **em 1980**"; "O exame realizou-se **a semana passada**"; "O Pedro está **sempre** a queixar-se".

- **Modificador de Fim** - Aplica-se a todas as strings que indiquem o objectivo ou propósito da acção descrita.
  Por exemplo: "O Pedro comprou um carro **para poder viajar ao fim-de-semana**".

- **Modificador de Modo** - Aplica-se a strings que especifiquem a forma/modo como uma acção é praticada ou decorre. Devem ser strings que respondam à pergunta "Como?"
  Por exemplo: "O Pedro falou **pausadamente**"; "A Maria caiu das escadas **de forma aparatosa**."

- **Modificação de Direcção** - Aplica-se a referências direccionais, podendo englobar tanto a "Fonte/Origem" como o "Destino" da deslocação.
  Por exemplo: "O comboio fez a primeira viagem **para o Alentejo**"; "O Pedro é natural **de Lisboa**"; "O Pedro deu um passo **em frente** e parou."

- **Modificador de Predicação Secundária** - Aplica-se manualmente em casos de predicados secundários (sobretudo com verbos no particípio passado), e a estruturas predicativas (resultativas, por exemplo)
  Por exemplo : "O Pedro trabalha na TAP **como comissário de bordo**"; "A Ana encontrou o assassino já **morto**";

- **Modificador de Ponto de Vista** - Aplica-se em strings que expressem posição ou ponto de vista do autor do enunciado. Não fazem parte da estrutura predicativa.
  Por exemplo: "Na minha opinião"; "a meu ver"

Considerando a Figura 1.1 com etiquetas gramaticais e semânticas, é possível observar que as palavras "João" e "carroça" denotam entidades relacionadas debaixo da relação denotada pelo verbo "comprou". Por outras palavras, o verbo da frase "comprou" define que a entidade João (primeiro argumento) adquiriu a entidade carroça (segundo argumento).

No fundo, as etiquetas semânticas definem as relações estabelecidas entre os vários constituintes da frase, ao nível do significado da mesma. Por outro lado, as etiquetas gramaticais ilustram relações sintácticas entre os constituintes, mas não possibilitam por si só, estabelecer relações semânticas entre os mesmos.

1.7.1 Etiquetas Gramaticais versus Etiquetas Semânticas

Um bom exemplo para mostrar que com um corpus que apenas contenha etiquetas gramaticais se torna mais difícil capturar a semântica de uma frase são as frases na voz activa

Etiqueta gramatical	**Significado**
ARGX	Argumento
ARGA	Agente causativo de verbos com alternância causativa
M-ADV	Modificador de Advérbio
M-MNR	Modificador de Modo
M-LOC	Modificador Locativo
M-PRED	Modificador de Predicação Secundária
M-TEM	Modificador Temporal
M-EXT	Modificador de Extensão
M-PNC	Modificador de Fim
M-CAU	Modificador de Causa
M-POV	Modificador de Ponto de Vista
M-DIR	Modificação de Direcção

Tabela 1.4: Tabela com as etiquetas semânticas presentes no corpus de dependências do Grupo NLX.

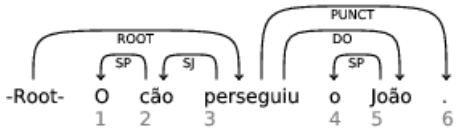


Figura 1.5: Anotação das relações gramaticais das palavras na frase na voz activa: "O cão perseguiu o João."

e na voz passiva.

Consideremos as frases na Figura 1.5 e Figura 1.6 anotadas com os arcos de dependências e respectivas etiquetas gramaticais (apenas).

Podemos observar, numa análise geral, que na frase na voz activa (Figura 1.5) a palavra "cão" é o *Sujeito* da frase, e que a palavra "João" é o *Objecto Directo* da mesma frase. No entanto, considerando a frase na voz passiva (Figura 1.6) que possui o mesmo significado que a frase na voz activa, podemos constatar que as palavras "cão" e "João" possuem relações gramaticais diferentes. Na frase na voz passiva, a palavra "João" é o *Sujeito* da frase, e a palavra "cão" é o *Complemento Oblíquo* da mesma frase. É importante referir

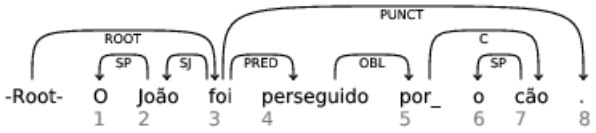


Figura 1.6: Anotação das relações gramaticais das palavras na frase na voz passiva: "O João foi perseguido pelo cão."

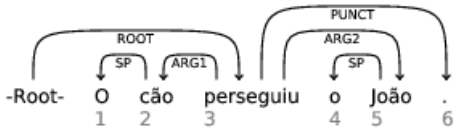


Figura 1.7: Anotação dos papéis semânticas das palavras na frase na voz activa: "O cão perseguiu o João."

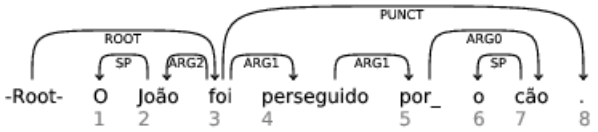


Figura 1.8: Anotação dos papéis semânticas das palavras na frase na voz passiva: "O João foi perseguido pelo cão."

que apesar da proposição "por" (Figura 1.6) depender do verbo "perseguido" com a etiqueta gramatical "OBL", esta preposição actua como um elo de ligação entre o verbo e a palavra "cão", sendo que "cão" como já referido, é o complemento oblíquo da frase.

Se considerarmos as mesmas frases com os arcos de dependências e respectivas etiquetas gramaticais e também semânticas, na voz activa (Figura 1.7) e na voz passiva (Figura 1.8), podemos ver que a palavra "cão" se relaciona com o verbo contendo a mesma etiqueta semântica (ARG1) em ambas as frases, e que a palavra "João" se relaciona com o verbo contendo a mesma etiqueta semântica (ARG2) em ambas as frases.

Com este tipo de etiquetas semânticas que definem os arcos de dependências entre as palavras de uma frase, é possível retirar a mesma informação semântica entre duas frases com ordem de palavras diferentes, mas possuidoras do mesmo significado.
Contudo interessa referir que apenas o corpus de dependências com as etiquetas gramaticais estava disponível aquando da busca por o melhor parser de dependências para a Língua Portuguesa. Acontece que o processo de adicionar as etiquetas semânticas ao corpus de dependências ainda decorria. Assim sendo, no próximo capítulo em que será explicado detalhadamente a busca pelo melhor parser de dependências para a língua portuguesa, o corpus de dependências usado para treinar e avaliar o resultado deste tipo de ferramentas, conterá os arcos de dependências definidos apenas com etiquetas gramaticais.

No Capítulo 4 em que será explicada a ferramenta de anotação semântica de paginas web (*Marcador Semântico*), o corpus de dependências usado já conterá os arcos de dependências definidos com as etiquetas gramaticais e semânticas.

1.8 Considerações Finais

Neste capítulo apresentou-se a "Motivação" e a "Contribuição" desta dissertação. Foram também explicados alguns conceitos que derivam da utilização de ferramentas de parsing de dependências, tais como a "Projectividade" e um formato de input aceite por este tipo de ferramentas, chamado de "CoNLL".

Explicou-se também a importância das etiquetas semânticas na secção *Relações Semânticas*.

No próximo capítulo, descrever-se-á os parsers de dependências encontrados, assim como o parser de dependências escolhido, que será usado para a concretização das ferramentas *LX Dep Parser* e *Marcador Semântico*.

Capítulo 2

Selecção do Parser de Dependências

2.1 Introdução

Uma vez que será necessário obter um grafo de dependências a partir de uma frase, para se poder extrair a semântica da mesma frase, efectuou-se uma busca por parsers de dependências. Efectuada a busca, deu-se início a um processo de escolha do melhor parser de dependências para a língua Portuguesa.

Os Parsers de Dependências que fossem treináveis com um corpus, que contivesse frases escritas na Língua Portuguesa anotadas com as dependências de cada palavra da respectiva frase, seriam os de maior interesse. Assim sendo, após uma pesquisa realizada, os Parsers de Dependências encontrados foram os seguintes:

- ISBN Dependency Parser
- MST Parser
- DeSR Dependency Parser
- Malt Parser
- KSDEP

O processo de busca realizado por parsers de dependências, teve como alvo a mailling list Corpora (mailling list que é frequentada por pessoas que pertencem à área de Processamento da Linguagem Natural[1]. Esta procura também fora efectuada nos motores de busca do Google e Yahoo, e também em repositórios como: Language Tecnology World,[2] que é um dos serviços mais abrangentes da World Wide Web, que possui um vasto reportório de tecnologias directamente envolvidas na área da Linguagem Humana.

[1] `http://gandalf.aksis.uib.no/corpora/`
[2] `beta.lt-world.org`

O repositório[3] Associação Internacional de Linguística Computacional, que envolve pessoas dentro da área da Linguagem Natural e Computacional, também foi outra fonte desta busca.

No processo de escolha do melhor parser de dependências para a Língua Portuguesa, as cinco ferramentas encontradas foram submetidas a um teste chamado de "Ten Fold Cross Validation" (Validação Cruzada de Dez Partições).
De referir que nem todos os parsers de dependências possuem ferramentas de avaliação, pelo que foi necessário encontrar um única ferramenta de avaliação, para avaliar o resultado do teste "Ten Fold Cross Validation".

Passemos então a descrever a ferramenta de avaliação utilizada assim como o teste "Ten Fold Cross Validation", sendo que depois seguir-se-á uma sucinta descrição sobre os parsers de dependências encontrados, de modo a revelar e esclarecer qual o parser de dependências que foi escolhido.

2.2 Malt Eval

Como método de avaliação dos parsers de dependências, foi utilizado um tipo de teste chamado de "Ten Fold Cross Validation" (Validação Cruzada de Dez Partições).

Neste tipo de testes (em que pode ser usado um qualquer número de partições para avaliacao, mas que por norma, são utilizadas dez partições), a amostra original dos dados é partida, sob um determinado critério, em dez partições. Deste modo, nove partições irão ser usadas como dados de treino pelo parser, enquanto a restante partição ira ser usada como dados de teste, também pelo parser. A análise/processamento do parser sobre esta ultima partição (corpus dourado), será comparada com a mesma partição guardada para ser analisada/processada pelo parser. Com esta comparação, e levando em conta as métricas descritas anteriormente, será calculada a taxa de acerto do parser, relativamente aos dados analisados/processados.
Este teste será repetido tantas vezes quanto o número de partições (dez vezes).

De modo a calcular a taxa de acerto dos vários parsers de dependências, seria ideal encontrar um avaliador único para os resultados de parsing dos vários parsers de dependências pois nem todos os parsers possuem um avaliador integrado.
Assim sendo, a ferramenta de avaliação Malt Eval foi utilizada para avaliar os resultados de todos os parsers.

Malt Eval é uma ferramenta freeware, escrita em Java, para avaliação de árvores de dependências que oferece várias métricas de avaliação e permite também a visualização das árvores de dependências. É uma ferramenta flexível na medida em que permite escolher um elevado número de parâmetros que são facilmente configuráveis

[3] `www.aclweb.org`

As métricas usadas para a avaliação dos parsers de dependências, cujo valores variam entre 0 e 1 (ou entre 0% e 100%), foram:

- **Unlabeled Attachment Score** (UAS) - Um token (símbolo/palavra) será contado com sucesso se o arco de dependências entre duas palavras da frase processada pelo parser de dependências estiver correcto em relação ao corpus dourado.
- **Labeled Attachment Score** (LAS) - Um token (símbolo/palavra) será contado com sucesso se o arco de dependências e a etiqueta semântica entre duas palavras da frase processada pelo parser de dependências estiverem correctas em relação ao corpus dourado.
- **Label Right** (LA) - Um token (símbolo/palavra) será contado com sucesso se a etiqueta semântica do arco de dependências entre duas palavra da frase processada pelo parser de dependências estiver correcto em relação ao corpus dourado. Esta métrica servirá de informação adicional uma vez que esta não é costume usar esta métrica em avaliações deste tipo de ferramentas.

As linhas de comandos para correr o Malt Eval são:

```
java -jar MaltEval.jar -s parsedSentences -g goldSentences
                        --Metric X
```

- **-s parsedSentences** - As frases cujos arcos de dependências e respectivas etiquetas semânticas foram calculadas pelo parser de dependências. Este ficheiro deverá estar em formato CoNLL 2006
- **-g goldSentences** - As frases cujos os arcos de dependências e respectivas etiquetas semânticas são anotadas à partida. Este ficheiro, em formato CoNLL 2006 será levado como base de comparação pela ferramenta Malt Eval.
- **–Metric X** - O tipo de métrica a ser usada na avaliação/comparação.

2.3 ISBN Dependency Parser

O ISBN Parser [21, 23] foi desenvolvido em parceria por Ivan Titov e James Henderson, da Universidade de Geneva e da Universidade de Edimburgo, respectivamente.

É um parser probabilístico que usa modelos probabilísticos baseados num histórico de decisões tomadas que prevê as derivações mais prováveis para a análise de dependências. O modelo probabilístico usado é um modelo baseado em “Incremental Sigmoid Belief Networks”.

Uma Belief Network (rede de crença) é um grafo acíclico dirigido que codifica dependências estatísticas entre duas varáveis. Cada variável S_i contida no grafo, tem associada uma probabilidade de distribuição

$$P(S_i|Par(S_i)) \tag{2.1}$$

sobre os seus valores, dados os valores dos seus parentes Par(Si) no grafo.

Uma "Sigmoid Belief Network" são "Bayesian Network" que possuem variáveis binárias e probabilidades de distribuição condicionais, na forma de uma função logística sigmóide:

$$P(S_i = 1|Par(S_i)) = \sigma(\sum_{S_j \in Par(S_i)} J_{ij}S_j) \tag{2.2}$$

onde S_i representa as variáveis, $Par(S_i)$ são as variáveis das quais depende S_i, σ denota a função sigmoid logística, e J_{i_j} será o peso para o arco que irá da variável S_i, para a variàvel S_j.

Para se usar as "Sigmoid Belief Network" de modo a que seja possível processar sequências longas de dados, tais como, uma sequência de decisões de uma parser ($D^1, \ldots, D^m$), as "Sigmoid Belief Network" são extendidas para uma forma de "Dynamic Bayesian Network".
Numa "Dynamic Bayesian Network", um novo conjunto de variáveis é instanciado para cada posição da sequência de decisões, mas os arcos e pesos dos arcos mantém-se inalterados em todas as posições da sequência.

"Incremental Sigmoid Belief Networks" diferem das "Sigmoid Belief Network" dinâmicas na medida em que permite que o modelo criado seja modificado (arcos e pesos dos arcos, por exemplo) incrementalmente, através da análise de cada decisão da sequência de decisões.

Para executar a função de parsing sobre uma determinada frase, o ISBN Parser utiliza um algoritmo que consiste numa pilha S que defini o estado corrente do parser e uma fila I que conterá as palavras do input que ainda estão por analisar, e a estrutura de dependência parcialmente construída, a partir de decisões anteriores construídas por decisões anteriores do parser.
O algoritmo começa com a pilha S vazia e termina quando a fila I ficar vazia. Este algoritmo utiliza quatro tipos de decisão:

- **Left-Arc**$_r$ - cria o arco de dependência entre a próxima palavra da fila (w_j) e a palavra w_i no topo da pilha, seleccionando a etiqueta `r` para o arco entre w_i e w_j. A palavra w_i é retirada (pop) da pilha.

- **Right-Arc**$_r$ - cria o arco de dependência entre a palavra no topo da pilha w_i, e a próxima palavra w_j na fila, seleccionando a etiqueta `r` para a relação/arco entre w_i e w_j.

- **Reduce** - retira do topo da pilha S, a palavra w_i
- **Shift**$_{w_j}$ - muda a palavra w_j da fila, para a pilha.

O modelo de probabilístico utilizado pelo ISBN Parser utiliza um modelo proposto em [13], utilizando parsing probabilístico que usa um método de "previsão de palavra", contido na acção "Shift" do parser. Esta predição da palavra é baseada em etiquetas morfo-sintácticas e em etiquetas gramaticais de granularidade fina, que são disponibilizadas no corpus de treino. Predição essa que começa pelas etiquetas gramaticais de granularidade fina da palavra desconhecida, passando pela etiqueta morfo-sintética, e finalizando com a própria palavra. Os autores do ISBN Parser afirmam que esta abordagem lhes permitirá diminuir o efeito de esparssez, evitando uma normalização (overfiting) das palavras no vocabulário.

O modelo probabilístico baseado em histórico, do ISBN Parser, decompõe a probabilidade de um parse de acordo com as decisões tomadas pelo parser, através da seguinte formula:

$$P(T) = P(D^1, \ldots, D^m) = \prod_t P(D^t | D^1, \ldots, D^{t-1}) \tag{2.3}$$

onde T é o parse e $D^1, \ldots, D^m$ é a sequência das decisões tomadas pelo parser.

Cada decisão D^t do ISBN Parser, para um determinado parse, pode ser dividida numa sequência de decisões elementares:

$$P(D^t | D^1, \ldots, D^{t-1}) = \prod_k P(d_k^t | h(t,k)) \tag{2.4}$$

onde $h(t,k)$ denota um histórico de decisões tomadas anteriormente.

Resumindo, desde que existe palavras a serem analisadas na fila I, para um dado estado S', é tomada uma sequência de decisões em que para cada decisão, são levadas em conta decisões tomadas anteriormente.
Decisões anteriores essas que são escolhidas consoante a estrutura de Dependencia (parse) que ate ao momento foi construída.
Esta nova sequência de decisões definida para o estado actual, levara o parser para um novo estado S'', que terminara a sua execução, caso não exista mais palavras na fila I.

De referir que após o processo de parsing, é utilizado uma variante do algoritmo "beam search" descrito em [22], de modo a determinar qual o parse mais provável. O algoritmo beam search é um algoritmo de busca heurística. É uma optimização do algoritmo de busca "best-first". O algoritmo de busca "best-first" é um grafo de busca que ordena todas as soluções parciais (estados) de acordo com uma heurística, na tentativa de prever qual a solução parcial que mais se aproxima da solução final (estado alvo). No algoritmo "beam search", apenas um número pré definido de melhor soluções parciais é mantido.

De seguida mostra-se resultados obtidos na conferência ”CoNLL-2007 shared task“. Estes resultados são o fruto de treinar o ISBN Parser e analisar o mesmo, com vários corpora de diversas linguagens. Conjuntos esses que possuem entre dois mil a cinco mil tokens.
De referir que nestas experiências foram utilizados “cortes de frequência’. Estes cortes de frequência servem para ignorar uma palavra, lema ou uma característica, que ocorra menos que o valor atribuído, para o corte de frequência. O corte de frequência com valor 20 foi usado para as línguas Chinesa e Grega. Para as restantes linguagens, o corte de frequência utilizado foi de 5, uma vez que os autores do ISBN Parser notaram que um corpus de treino com um maior número de palavras, lemas e características de palavras, diminuíam a eficiência do parser.
A tabela seguinte demonstra os resultados de avaliação com corpus de várias línguas, na conferência CoNLL de 2007:

Língua	LAS	UAS
Árabe	0,7410	0,8320
Basco	0,7550	0,8190
Catalão	0,8740	0,9340
Chinês	0,8210	0,8790
Checo	0,7790	0,8420
Inglês	0,8840	0,8970
Grego	0,7350	0,8120
Húngaro	0,7790	0,8220
Italiano	0,8230	0,8630
Turco	0,7980	0,8620

2.3.1 Execução

Dos cinco parsers de dependências aqui apresentados, este será o que apresenta menos facilidades de interacção.
Para o correcto funcionamento deste parser, é necessário executar uma série de passos (de notar que os comandos apresentados de seguida são executados em Linux):

- `./prepare_data FREQ_CUTOFF UNKN_FREQ_CUTOFF PROJECT_PATH TRAINING_FILE VALIDATION_FILE OTHER_FILES`:
  Este comando preparará uma directoria onde todos os ficheiros que o parser precisa para treinar e analisar serão colocados:
  - **PROJECT_PATH** a directoria a ser criada onde serão gerados ficheiros de “configuração” para treino e parsing.
  - **FREQ_CUTOFF** é um inteiro que indica que caso uma palavra, lema, etiqueta morfo-sintáctica, no conjunto de treino, ocorra em menor número de vezes que `FREQ_CUTOFF`, será ignorado.

  - **UNKN_FREQ_CUTOFF** é também um inteiro caso um item desconhecido (palavra, lema, etiqueta-morfo-sintáctica) ocorra, no corpus de treino, em menor número de vezes do que o `UNKN_FREQ_CUTOFF`, este item será "fundido" com outro item menos frequente, da mesma categoria.
  - **TRAINING_FILE** representa o ficheiro (corpus) em formato CoNLL 2006, que será usado para treinar o parser. O ficheiro indicado por `TRAINING_FILE` será convertido para o formato CoNLL.ext.
  - **VALIDATION_FILE** é um sub-conjunto do TRAINING_FILE que deverá conter pelo menos dois mil tokens.
  - **OTHER_FILES são** outros ficheiros em formato CoNLL 2006, a serem eventualmente analisados pelo parser, que serão convertidos para o formato CoNLL.ext.

- Num segundo passo, é necessário assegurar que o tamanho das estruturas corresponde aos parâmetros do treebank, ou seja, é necessário configurar no ficheiro idp_io_spec.h, que está na directoria `PROJECT_PATH`, alguns campos como, por exemplo, MAX_CPOS_SIZE, pois se treinarmos este parser com o corpus de dependências do Grupo NLX, iremos ter mais do que trinta (valor definido por defeito) etiquetas morfo-sintácticas (Part-of-Speach Tags).

  Será necessário também copiar os ficheiros parser.par, parser.ih e parser.hh da directoria sample para a directoria PROJECT_PATH. Também é necessário indicar no ficheiro parser.par, o correcto caminho dos ficheiros de treino e validação criados pelo script `./prepare_data`.

- De seguida poder-se-á treinar este parser executando o comando:

  ```
  ./idp -train PROJECT_PATH/parser.par
  ```

  onde se chama o executável "idp", com a opção "-train", indicando o ficheiro de configuração parser.par.

- No passo seguinte é necessário converter o(s) ficheiro(s) de teste (OTHER_FILES), que serão analisados (parsing), para o formato CoNLL.ext. Formato esse que não será mais do que um formato CoNLL 2006 com valores numéricos afectos a cada palavra de modo a serem usados pelo parser, no processo de parsing.
  O utilizador pode fazer a converção dos ficheiros a analisar logo no início da criação da directoria com o script prepare_data ou então utilizando o script:

  ```
  conll2ext PROJECT_PATH TRAINING_FILE.conll
            FILE_TO_CONVERT.conll
  ```

onde:

  - **PROJECT_PATH** é a directoria que fora criada com o script prepare_data.
  - **TRAINING_FILE.conll** é o mesmo ficheiro de treino que fora usada na criação da directoria PROJECT_PATH.
  - **FILE_TO_CONVERT.conll** será o ficheiro que queremos fornecer ao parser. Este ficheiro será convertido para o formato conll.ext, para posteriormente ser analisado.

- De seguida podemos proceder ao parsing utilizando o comando:

```
./idp -parse PROJECT_PATH/parser.par TEST_FILE OUT_FILE:
```

  - **PROJECT_PATH/parser.par** o ficheiro de configuração presente na directoria PROJECT_PATH.
  - **TEST_FILE** será o ficheiro para analisar convertido para o formato CoNLL.ext no passo anterior.
  - **OUT_FILE** é o resultado do parser após parsing do ficheiro TEST_FILE.

- O próximo passo será o de converter o resultado do parser (já com os arcos de dependência e respectivas etiquetas semânticas calculadas). Para tal utilizar-se-á o comando:

```
ext2conll PROJECT_PATH test.conll parser_res.conll.ext
                parser_res.conll.proj
```

  - **PROJECT_PATH** directoria criada.
  - **parser_res.conll.ext** o resultado do parser.
  - **parser_res.conll** o resultado do parser em formato CoNLL 2006, com projectividade.

- Para terminar este ISBN parser vem acompanhado de um avaliador que corre com o seguinte comando:

```
./eval07.pl -g gold_std.conll -s parser_res.conll
```

  - **-g gold_std.conll** o corpus dourado em formato CoNLL 2006.
  - **-s parser_res.conll** o ficheiro analisado e reconvertido para o formato CoNLL 2006.

2.3.2 Avaliação

Após uma experimentação com este parser e seguindo todas as instruções, não foi possível obter resultados com este parser de dependências. Começando pela interface (entenda-se por comandos inseridos de modo a correr a ferramenta) esta não é de todo intuitiva, sendo o parser, dentro de estes cinco parsers recolhidos, o mais difícil de trabalhar. São precisos muitos passos até se conseguir efectuar o parsing, desde criar uma directoria com ficheiros de treino e teste, até desconverter o resultado de parsing (de conll.ext para conll) para serem avaliados. Por outras palavras, certos passos na execução de este parser deveriam ser "fundidos" para facilitar a tarefa ao utilizador. Para além da conversão dos ficheiros de treino e teste em formato CoNLL para um formato CoNLL.ext, é necessário criar ficheiros de validação para o treino desta ferramenta. Como já fora descrito, estes ficheiros de validação deverão conter no mínimo dois mil símbolos (tokens), ou seja, frases em formato CoNLL 2006 do ficheiro de treino que perfaça os dois mil tokens.
Também se pode colocar o próprio ficheiro de treino como ficheiro de validação, mas isso leva-nos a um treino com uma duração a rondar as cinco horas [4].

O mais importante é que a "robustez" do funcionamento desta ferramenta não parece ser a melhor, pois o ficheiro de teste que será analisado pelo parser não deverá conter etiquetas morfo-sintácticas que não ocorram no ficheiro de treino, pois caso contrário a preparação/conversão (com os scripts ./prepare ou ./convert) não terminará com sucesso.
Mesmo corrigindo esta questão são encontrados problemas durante a execução do parser ao efectuar o parsing sobre os ficheiros de teste, não sendo possível que se conclua com sucesso o parsing.

2.4 KSDEP / LRDEP

O parser de dependências probabilístico KSDEP [24, 20], é um parser desenvolvido por Kenji Sagae e Jun'ichi Tsujii, das universidades de Tokyo e Manchester, respectivamente.

O KS/LR Dep Parser define-se como uma variante do algoritmo de parsing LR, para parsing de dependências, sendo também aplicado um algoritmo de procura "best-first" de modo a ir ao encontro da generalização do parsing probabilístico de dependências.

Um algoritmo de parsing LR, é um parser que lê o input da esquerda para a direita, produzindo um resultado final chamado de "Rightmost derivation", baseado numa dada gramática. Um LR parser é baseado num algoritmo que tem como base de decisão uma tabela de parsing (parser table), que é uma estrutura de dados que contém informação sintáctica sobre a linguagem, que se encontra a ser parsada. Como tal, o termo LR parser diz respeito a uma classe de parsers que possuem a capacidade de processar quase todas as linguagens, desde que seja fornecida uma tabela de parsing, gerada por um "parser generator".

[4] Experiência efectuada com o corpus de dependências do Grupo NLX.

O parsing do algoritmo LR, comparando com outros algoritmos de parsing como por exemplo o LL parsing, consegue manipular um maior número de línguas e consegue também uma melhor descrição sobre erros, ou seja, entre vários erros que possam surgir, o algoritmo LR consegue detectar erros de sintaxe, quando o input não corresponde à gramática, com a maior celeridade possível, contrastando com o algoritmo LL que devido ao backtracking, torna a localização do erro de sintaxe, bastante mais difícil.

Voltando ao KS Dependency Parser, este parser como já fora referido, utiliza uma variante do algoritmo LR, entendida por um algoritmo "best-first". A variante do LR parser não utiliza uma tabela de parsing para determinar qual o passo a tomar, no processo de parsing. Ao invés, é utilizado um classificador para determinar as acções de mudança e redução, com informação derivada do input. Informação essa que também estaria presente numa tabela de parsing (os ficheiros no topo da pilha, e os restantes itens da string de input). A variante do algoritmo LR que constitui o KS Dependency Parser, funciona com base em duas estruturas de dados: uma pilha S que contém sub-árvores da árvore de dependências final, para um dado input, e uma fila Q que contém as palavras de um dado input. De referir que S é inicializado sem nenhum valor, e Q é inicializado com as palavras do input.

Este algoritmo executa duas acções principais: mudança (shift) e redução (reduce). Quando uma acção de mudança é concretizada, uma palavra é transferida do início da fila Q, para o topo da pilha S (representando uma árvore com apenas um nó, a própria palavra). Quando uma acção de redução é efectuada, os dois elementos no topo da pilha S (s_1 e s_2), são extraídos e um novo item é colocado em S. Este novo item representa o arco de dependência entre s_1 e s_2 e respectiva etiqueta.
O parsing termina quando a fila Q estiver vazia (ou seja, quando todas as palavras tiverem sido processadas), e a pilha S contiver apenas uma árvore (a árvore final de dependências). Se a fila Q estiver vazia, e a pilha S não estiver vazia e não for possível realizar mais acções de redução, o parsing termina e o input é rejeitado.

Com este modelo determinístico descrito anteriormente, os autores do KS DEP procedem a uma extensão da variante do algoritmo LR acima descrita, tornando o KSDEP um parser probabilístico, por outras palavras, ao invés do classificador retornar uma acção para o parser executar, o classificador retorna uma série de acções a tomar com as correspondentes probabilidades, sendo que a probabilidade de uma árvore como resultado de parse, é o produto das probabilidades das acções tomadas aquando a sua derivação.

De modo a encontrar o parse mais provável de acordo com o modelo probabilístico LR, é necessário uma estratégia de "best-first", que implica uma extensão do algoritmo determinístico descrito anteriormente.

Esta extensão é efectuada da seguinte maneira: uma nova estrutura de dados T_i é criada.

T_i representa um estado do parser que contém uma pilha S_i, uma fila Q_i e uma probabilidade P_i. Com o algoritmo "best-first", é criada uma "heap" H que contém vários estados do parser ($T_0 \ldots T_m$).
Estes estados estão ordenados em H de acordo com a probabilidade de cada um. H é inicializado de modo a conter um estado T_0 que contém: uma pilha S_0, uma fila Q_0 e uma probabilidade associada $P_0 = 1.0$.
O algoritmo "best-first" entra num ciclo que só termina quando H estiver vazio. A cada iteração é obtido o estado $T_{current}$ a partir de H. Se $T_{current}$ corresponde ao estado final (onde $Q_{current}$ é vazio e $S_{current}$ contém apenas um item), é retornado o item de $S_{current}$ que consiste na estrutura de Dependencia da frase de input. Se $T_{current}$ não corresponder ao estado final, é obtido uma lista de acções do parser ($act_0 \ldots act_n$) em que cada acção contém uma probabilidade associada ($Pact_0 \ldots Pact_n$).
Para cada acção act_j do parser na lista obtida, é criado um novo estado do parser T_{new} aplicando act_j a $T_{current}$, ajustando a probabilidade de T_{new} ($P_{new} = P_{current} * Pact_j$).
No final, o estado T_{new} é inserido em H. Assim que cada novo estado criado a partir de cada acção do parser, tiver sido inserido em H, o algoritmo avança para a próxima iteração.

Para a conferencia CoNLL 2007, os autores de KS Dependency Parser, treinaram três modelos LR com o KSDEP Parser, em que cada input é analisado usando os três modelos LR individualmente, obtendo como resultado três estruturas de dependências para um dado input que são depois combinadas para formar a árvore/grafo de dependências final. Esta combinação das estruturas de dependências resultantes dos três modelos LR, é feita através de acordo com o esquema de combinação "maximum-spanning-tree", onde cada dependência proposta por cada um dos três modelos possui o mesmo peso.
Dos três modelos LR treinados, o primeiro foi treinado com um classificador de entropia máxima para determinar as acções a tomar pelo parser e suas probabilidades, o segundo usou também o classificador baseado em entropia máxima, mas com o parsing realizado de "trás-para-frente", ou seja, a string de input é invertida antes de se proceder ao parsing, pois observado que o procedimento de combinar vários parsers finais, poderá ser benéfico.
O último modelo descrito, foi treinado com o classificador support vector machines. De referir que os autores treinaram este último modelo com o classificador em modo determinístico, uma vez que não foi observado melhorias nos resultados finais (taxas de acerto) com o classificador support vector machines em "modo" probabilístico.
A seguinte Tabela representa os resultados obtidos no conferencia CoNLL de 2007, com a combinação dos três modelos LR, para as dez línguas diferentes:

Língua	LAS	UAS
Árabe	0,7471	0,8404
Basco	0,7464	0,8119
Catalão	0,8816	0,9334
Chinês	0,8469	0,8884
Checo	0,7483	0,8127
Inglês	0,8901	0,8987
Grego	0,7358	0,8351
Húngaro	0,7953	0,8351
Italiano	0,8391	0,8768
Turco	0,7591	0,8272

2.4.1 Execução

De modo a correr este parser será necessário treiná-lo para um modelo ser criado e usado no parsing de frases em formato CoNLL 2006.
Para se proceder ao treino deste parser, deve-se executar a seguinte linha de comando:

```
./ksdep -t TRAIN_FILE
```

- -t opção para treinar.
- TRAIN_FILE o ficheiro para treinar o parser, em formato CoNLL 2006.

Já o comando para analisar será o seguinte:

```
./ksdep -m MODEL_FILE INPUT_FILE
```

- -m opção para carregar um modelo previamente treinado.
- MODEL_FILE o modelo a ser carregado.
- INPUT_FILE o ficheiro a ser analisado, em formato CoNLL 2006.

Também existem outras opções dísponiveis a serem usados com o executável ./ksdep:

- -i define o parâmetro de regularização (valores inteiros menores que 1.0 poderão provocar “overfit”)
- -m esta opção a ser usada no treino define o nome do modelo
- -b define a “beam width”. Para efectuar um parsing determinístico basta utilizar o valor 1.
- -it define o número de iterações para o treino. A cada cem iterações é guardado para o disco uma versão do modelo de treino.

2.4.2 Avaliação

Os resultados que são de seguida apresentados, foram obtidos com o corpus de dependências NLX (apenas com etiquetas gramaticais):

Partição	LAS	LA	UAS
1	0,8770	0,9060	0,9230
2	0,8670	0,8920	0,9130
3	0,8450	0,8750	0,8960
4	0.8220	0,8650	0,8730
5	0,8530	0,8860	0,8930
6	0,8250	0,8720	0,8750
7	0,8580	0,8920	0,9010
8	0,8370	0,8770	0,8850
9	0,8780	0,9040	0,9110
10	0,8390	0,8720	0,8840
Média	**0,8501**	**0,8841**	**0,8954**

2.5 DeSR Dependency Parser

O DeSR Dependency Parser [3, 2, 7] é um parser de dependências desenvolvido por Giuseppe Attardi, Felice Dell'Orletta e Maria Simi do departamento de informática de Pisa, em Itália. Também foi desenvolvido por Atanas Chanev, da Universidade de Trento em Itália, e por Massimiliano Ciaramita, do centro de investigação da Yahoo, situado em Barcelona, Espanha. Este Parser de dependências implementa um algoritmo de parsing incremental determinístico "shift-reduce", usando regras específicas para lidar com dependências não projectivas, o que permite que o parsing de uma frase seja determinístico. As estruturas de dependências são construídas analisando o input da esquerda para a direita e decidindo a cada passo a realização de uma acção de "mudança" ou a criação de uma dependência entre dois tokens adjacentes. Este parser utiliza um classificador para aprendizagem e predição da próxima acção a ser tomada pela parser.

De modo a exemplificar/demonstrar o funcionamento do DeSR parser, os estados do parser podem ser definidos como quadruplos $< S, I, T, A >$ onde S é a pilha dos tokens analisados, I é a lista dos tokens que permanecem por analisar, T é uma pilha de tokens temporários e A é o arco de relação para o grafo de dependências.
Dado uma string W como input, o estado inicial do parser será $< (), W, (), () >$, e termina com $< S, (), (), A >$. As três regras básicas usadas por este parser para analisar uma frase, são:

- Shift$_d$: $< S, n|I, T, A > \rightarrow < n|S, I, T, A >$
- Right$_d$: $< s|S, n|I, T, A > \rightarrow < S, n|I, T, A \cup (s, d, n) >$

- Left$_d$: $< s|S, n|I, T, A > \rightarrow < S, s|I, T, A \cup (n, d, s) >$

De referir que as regras de Left e Rigth são usadas para etiquetar e juntar tokens numa relação de dependência. A cada passo, o parser usa classificadores treinados sobre um corpus de modo a conseguir prever que acção o parser deve tomar e que etiqueta deve ser atribuída ao arco de dependência entre dois tokens.

Contudo, para lidar com arcos de dependências não projectivos é usada uma técnica proposta por [17].
O parser DeSR usa as seguintes regras para analisar uma dependências não projectivas:

- Right2$_d$: $< s_1|s_2|S, n|I, T, A > \rightarrow < S, s|I, T, A \cup (s, d, n) >$
- Left2$_d$: $< s|S, n|I, T, A > \rightarrow < S, s|I, T, A \cup (n, d, s) >$
- Right3$_d$: $< s_1|s_2|S, n|I, T, A > \rightarrow < S, s|I, T, A \cup (s, d, n) >$
- Left3$_d$: $< s|S, n|I, T, A > \rightarrow < S, s|I, T, A \cup (n, d, s) >$
- Extract : $< S, n|I, T, A > \rightarrow < n|S, I, T, A >$
- Insert : $< S, n|I, T, A > \rightarrow < n|S, I, T, A >$

As regras $Left2$, $Right2$ são semelhantes ás regras $Left$ e $Right$, com a excepção de que estas regras criam ligações entre nós, usando um nó intermediário, enquanto que as regras $Left3$ e $Right3$ criam ligações entre nós usando dois nós intermediários. As acções $Extract/Insert$ generalizam as acções anteriores, movendo uma token para a pilha T e re-inserindo o topo de T em S.

Este parser pode utilizar um dos seguintes algoritmos para aprendizagem:

- **Maximum Entropy** - O principio da máxima entropia é útil quando aplicado a informação que seja testável. Uma informação é testável se for possível determinar que uma dada distribuição é consistente com essa mesma informação.
  A Entropia Máxima procura a probabilidade de distribuição que maximize a entropia (medida de incerteza associada com um valor aleatório), de uma dada informação.
  O principio da máxima entropia é também usado para criar modelos de dados. Dados esses que se assumem que é informação testável. Estes modelos são usados na área do Processamento da Linguagem Natural.

- **Multilayer perceptron** - é um modelo baseado em redes neurais artificias que mapeia conjuntos de dados num dado output. É um tipo de rede neural que utiliza três ou mais camadas de neurões (nós) com funções de activação não linear. É bastante eficiente na medida em que consegue distinguir dados que são linearmente separáveis.

- **Memory-Based Learning** - Em aprendizagem automática instance-based learning ou memory-based learning fazem parte de uma família de algoritmos que comparam informação de input com informação vista no treino que está armazenada em memória. A "aprendizagem baseada em memória" faz parte de um tipo de aprendizagem chamada de "lazy learning".

- **Support Vector Machines** - são um conjunto de métodos de aprendizagem usados para classificação e regressão. O algoritmo Support Vector Machines, para um dado conjunto de dados para treino cria um modelo e dois tipos de categorias, e prevê se um novo exemplo se ajusta dentro de uma categoria ou outra categoria.

  Um modelo SVM consiste numa representação dos exemplos (informação) em pontos no espaço mapeados de modo a que os exemplos das duas categorias estejam divididas por uma lacuna bem visível. Os novos exemplos são então mapeados no espaço de pontos, conforme o modelo vá prevendo a categoria dos novos exemplos, sendo estes mesmos também representados por pontos.

- **Logistic Regression** - A regressão logística é usada para prever a probabilidade de ocorrência de um evento, sendo um modelo deste tipo utilizado para a predição da probabilidade de ocorrência de um evento por ajuste dos dados a uma curva logística. Este tipo de modelos utiliza variáveis de predição que tanto podem ser numéricas ou de categorias. A título de exemplo temos o cálculo de probabilidade de uma pessoa ter um ataque cardíaco dentro de um determinado período de tempo que se baseia na idade da pessoa, sexo e índice de massa corporal.

Na conferência CoNLL-X SharedTask 2006 em Nova Iorque, este parser de dependências entrou em competição tendo apresentado os seguintes resultados

	ME		MBL	
Língua	LAS	UAS	LAS	UAS
Árabe	0,5412	0,6950	0,5970	0,7469
Bulgaro	0,7290	0,8524	0,7917	0,8592
Chinês	0,7000	0,8133	0,7217	0,8308
Checo	0,6210	0,7344	0,6920	0,8022
Dinamarquês	0,7172	0,7884	0,7613	0,8365
Holandês	0,6371	0,6893	0,6897	0,7473
Alemão	0,7588	0,8025	0,7979	0,8431
Japonês	0,7801	0,8205	0,8339	0,8673
Português	0,7940	0,8503	0,8097	0,8678
Esloveno	0,6063	0,7214	0,6267	0,7660
Castelhano	0,7033	0,7425	0,7437	0,7990
Suéco	0,7520	0,8303	0,7485	0,8373
Turco	0,4883	0,6525	0,4758	0,6525

É necessário referir que segundo [2] foi experimentada a utilização de alguns algoritmos de aprendizagem com a finalidade de afinar a ferramenta. Um desses algoritmos de aprendizagem utilizados foi o SVM (Support Vector Machine), mas sem sucesso, pois só o treino para corpus pequenos como o português levou quatro dias, produzindo um modelo cuja sua utilização não era possível.

2.5.1 Execução

O comando que serve para treinar o parser é:

```
./desr -c desr.conf -t -m MODEL_FILE TRAIN_FILE
```

- -c desr.conf: o ficheiro de configuração onde entre outras coisas, é possível indicar o algoritmo de aprendizagem a utilizar, o formato de input e output dos dados.
- -t: opção que indica que o parser deverá executar as funções de treino/aprendizagem, utilizando o algoritmo de aprendizagem indicado no ficheiro de execução.
- -m MODEL_FILE: o nome do modelo de treino a ser criado.
- TRAIN_FILE: O ficheiro de treino a ser usado, em formato CoNLL 2006.

Após o treino, para analisar (parsing) frases basta utilizar o seguinte comando:

```
./desr -c desr.conf -m MODEL_FILE TEST_FILE > PARSED_FILE
```

- -c desr.conf: o ficheiro de configuração onde entre outras coisas, é possível indicar o algoritmo de aprendizagem a utilizar, o formato de input e output dos dados.
- -m MODEL_FILE: o nome do modelo de treino a ser utilizado.
- TEST_FILE: O ficheiro de treino a ser usado, em formato CoNLL 2006 (por defeito).
- PARSED_FILE: O ficheiro onde serão colocados os resultados (as frases com as respectivas dependências e etiquetas gramaticais associadas) do parser.

2.5.2 Avaliação

Com a experimentação deste parser, usando o corpus de dependências do NLX com etiquetas gramaticais, apenas foi possível obter resultados com os algoritmos MLP (Multilayer perceptron) e ME (Maximum Entropy). O algoritmo SVM não produziu resultados finais. O algoritmo SVM devolvia como output, ficheiros de parsing vazios. Também foi tentada a instalação de algoritmos de aprendizagem indicados pelos autores desta ferramenta (TiMBL e Snow), mas sem nenhum sucesso, mesmo tendo sido seguidos todos os passos. Os resultados obtidos foram os seguintes:

MBL:

Partição	LAS	LA	UAS
1	0,8820	0,9160	0,9240
2	0,8610	0,8990	0,9100
3	0,8810	0,9120	0,9140
4	0.8170	0,8740	0,8650
5	0,8710	0,9130	0,9000
6	0,8350	0,8780	0,8700
7	0,8880	0,9200	0,9120
8	0,8330	0,8750	0,8750
9	0,8800	0,9140	0,9120
10	0,8490	0,8850	0,9010
Média	**0,8597**	**0,8986**	**0,8983**

ME:

Partição	LAS	LA	UAS
1	0,882	0,916	0,924
2	0,861	0,899	0,91
3	0,881	0,912	0,914
4	0.817	0,874	0,865
5	0,871	0,913	0,9
6	0,835	0,878	0,87
7	0,888	0,92	0,912
8	0,833	0,875	0,875
9	0,88	0,914	0,912
10	0,849	0,885	0,901
Média	**0,8597**	**0,8986**	**0,8983**

2.6 MST Parser

O Maximum Spaning Tree Parser [19, 18] é um parser de dependências que foi desenvolvido por Ryan McDonald, Kevin Lerman e Fernando Pereira, no departamento de Ciência da Computação e Informação, Universidade de Pensilvânia.
Este ferramenta é um parser de dependências de duas fases. A primeira fase consiste no parsing de dependências sem etiquetação. Já a segunda fase consiste em pegar no resultado da 1ª fase e etiquetar todos os núcleos e dependentes que constituem as dependências, usando um classificador.

Consideremos uma frase dada como input caracterizada por:

$$x = x_1, \cdots, x_n \tag{2.5}$$

contendo n palavras, sendo que y corresponde ao grafo de dependências da frase x. Um grafo de dependências é representado por um conjunto de pares ordenados:

$$(i, j) \in y \tag{2.6}$$

em que x_j é o dependente e x_i é o núcleo da dependência, sendo criado o arco de dependência entre as duas palavras ($X_j e X_i$). A este arco de dependência é associado uma etiqueta l(i,j).

Na primeira fase são calculados os arcos de dependências entre duas palavras, sendo atribuída informação/características a esses mesmo arcos. Esta ferramenta utiliza um algoritmo de aprendizagem MIRA, cuja principal característica é definir um conjunto rico de informação/características através das decisões de parsing bem como níveis de informação/características relativos a essas mesmas decisões de parsing.
Para além do cálculo dos arcos de dependências, também é obtida informação morfo-sintáctica proveniente de cada palavra/token, ou seja, consideremos o seguinte arco de dependência:

Sendo a palavra "A" a cabeça da dependência e "B" o dependente, é adicionada informação morfo-sintáctica a estes constituíntes da dependência, A' B'. Esta informação permite que seja possível a modelação das consistências e semelhanças entre a cabeça e o dependente, em termos de género, caso ou número. No entanto nem todos os corpora de linguagens possuem esta informação, pelo que a informação morfo-sintáctica só será adicionada aos constituintes de uma dependência se estiver disponível.

Numa segunda fase, o parse de output y da frase x é alvo do cálculo das etiquetas sintácticas para cada arco de dependência $(i, j) \in y$. Cada arco terá então uma etiqueta $l_{(i,j)}$. Uma vez que os autores do MST Parser não conseguem colocar estas duas fases num só processo, de modo a resolver certas ambiguidades que eventualmente pudessem surgir, tratam o output da primeira fase, como um input da segunda fase, para cada arco de dependências $(i, j) \in y$ da frase x, tentando achar a etiqueta para o arco de dependência com o "score" mais alto, através da seguinte fórmula:

$$l_{(i,j)} = \arg\max_{\bar{l}}(l, (i, j), y, x) \tag{2.7}$$

que aplicada a cada arco de dependências, produz o output final. Contudo, os autores desta ferramenta acharam vantajoso que, aquando o cálculo das etiquetas gramaticais, se conheça as etiquetas gramaticais, dos arcos de dependências adjacentes. Por exemplo, para uma palavra x_i, com os seus dependentes (filhos) $x_j 1$,...,$x_j M$, acontece que geralmente muitos dos arcos de dependências possuem etiquetas gramaticais correlacionadas. Como tal, esta informação ((i,j_1),...,(i,j_M)) é modelada como uma sequência de

etiquetação: falta o traço sobre o L

$$(l_{(i,j_1)}, ..., l_{(i,j_M)}) = \bar{l} = \arg\max_{\bar{l}} s(l, i, y, x) \tag{2.8}$$

sobre o qual é aplicada uma factorização Markov de primeira-ordem:

$$\bar{l} = \arg\max_{\bar{l}} \sum_{m=2}^{M} s(l_{i,j_m}, l_{i,j_{m-1}}, i, y, x) \tag{2.9}$$

em que cada factor é o resultado (score) da etiquetação da dos arcos de dependências adjacentes (i,j$_m$) e (i,j$_{m-1}$), na árvore y.

Assim, esta factorização de primeira ordem pode ser entendida como um histórico da função de parsing, que é consultada de modo a melhorar a performance do MST Parser. Os autores também tentaram a aplicação de uma factorização de maior ordem (levando em conta mais arcos de dependências para além do arco de dependência actual e o imediatamente anterior), mas não foram verificadas quaisquer tipos de melhorias.

De modo a calcular o "score" para a etiqueta corrente, é utilizado um produto factorial entre as representações das características de cada palavra e um vector de pesos:

$$s(l_{(i,j_m)}, l_{(i,j_{m-1})}, i, y, x) = w \cdot f(l_{(i,j_m)}, l_{(i,j_{m-1})}, i, y, x) \tag{2.10}$$

Assumindo que existe a correcta representação das características das palavras, a etiqueta com o maior resultado (score) é encontrada com o algoritmo Viterbi. O algoritmo Viterbi é um algoritmo de programação dinâmica concebido por Andrew Viterbi em 1967, que tem a particularidade de, entre outras funções, conseguir devolver uma representação em string, a partir de um sinal acústico (reconhecimento de voz).
Este algoritmo encontra a sequência mais provável de estados escondidos, chamado de "caminho de Viterbi", que é fruto de uma sequência de estados observados. O algoritmo de Viterbi cria um número de pressupostos:

- Tanto os eventos observados como os eventos escondidos devem estar compostos em sequências ordenadas por tempo.
- As duas sequências que contêm os eventos observados e os eventos escondidos devem estar alinhadas, pois um instância de um evento observado deve corresponder a uma instância de um evento escondido.
- A computação da sequência mais provável de eventos escondidos até a um certo ponto t, deve depender apenas do evento observado no ponto t, e a sequência mais provável, no ponto t-1.

Todas estes pressupostos devem ser satisfeitos a partir de um modelo Markov de primeira ordem.

Este parser utiliza dois algoritmos de parsing (projectivo e não-projectivo) durante a fase de treino:

- **Chu–Liu/Edmonds's algorithm** (não-projectivo) - é um algoritmo para encontrar o máximo ou o mínimo caminho num dado grafo. Assumindo que os nós, num dado grafo, estão ligados por arcos, constituindo pares de nós, um algoritmo minimum spanning tree não pode ser usado, Como tal, é aplicado o algoritmo Chu and Liu (1965) e depois o algoritmo Edmonds (1967).

  Para encontrar o caminho de maior comprimento, o arco com o maior valor é encontrado e ligado entre os dois nós. Depois o próximo arco com o maior valor é encontrado e processado da mesma maneira. Se um arco criar um ciclo, esse arco é apagado. O caminho mínimo num grafo é calculado olhando para o arco com o menor valor.

- **Eisner algorithm** (projectivo) - é um algoritmo de programação dinâmica. Com este algoritmo, Eisner introduziu um modelo para efectuar parsing de dependências baseado no algoritmo CKY. O algoritmo CKY, tal como todos os algoritmos baseados em programação dinâmica, necessita um factorização do espaço de busca que permita a definição recursiva de sub-problemas.

Os resultados obtidos com esta ferramenta, na conferência "CoNLL-X 2006 Shared Task", nos quais se encontra uma avaliação efectuada com um corpus contendo frases anotadas em Língua Portuguesa:

Língua	LAS	UAS
Árabe	0,6690	0,7930
Bulgaro	0,8760	0,9200
Chinês	0,8590	0,9110
Checo	0,8020	0,8730
Dinamarquês	0,8480	0,9060
Holandês	0,7920	0,8360
Alemão	0,8730	0,9040
Japonês	0,9070	0,9280
Português	0,8680	0,9140
Esloveno	0,7340	0,8320
Castelhano	0,8230	0,8610
Suéco	0,8250	0,8890
Turco	0,6320	0,7470

Numa pequena nota, é necessário referir que os resultados obtidos para os corpus das seguintes línguas: Checo, Dinamarquês, Alemão, Holandês, Japonês, Português e Esloveno, foi utilizado o algoritmo de parsing não projectivo Chu–Liu/Edmonds's. Para as restantes linguagens foi utilizado o algoritmo de parsing projectivo Eisner.

2.6.1 Execução

Para se proceder ao treino desta ferramenta, será necessário executar o seguinte comando:

```
train train-file:trainFile model-name:dep.model
                decode-type:proj
```

em que:

- **train** - flag que indica à ferramenta para iniciar o treino sobre um dado corpus.
- **train-file:trainFile** - o corpus de treino *trainFile* que será fornecido para efectuar o treino do parser. Deverá estar no formato CoNLL 2006.
- **model-name:dep.model** - o nome a atribuir ao modelo de treino, depois de efectuar o treino do parser.
- **decode-type:proj** - o algoritmo (algoritmo projectivo Eisner) de parsing a ser usado no treino do parser (por omissão, o algoritmo de parsing a ser usado). Para efectuar o treino em modo não projectivo Chu–Liu/Edmonds's, em vez de *proj*, seria *non-proj*.

De modo a conseguir calcular os arcos de dependências e respectivas etiquetas gramaticais (parsing), é necessário correr o seguinte comando:

```
test model-name:dep.model test-file:testFile
           output-file:outputFile
```

onde:

- **test** - flag que indica à ferramenta para iniciar o parsing sobre o ficheiro de teste (testFile).
- **model-name:dep.model** - o modelo criado no fase de treino que será usado para efectuar a tarefa de parsing.
- **test-file:testFile** - o ficheiro que contém as frases que serão alvo da tarefa de parsing. Este ficheiro deverá estar em formato CoNLL 2006.
- **output-file:outputFile** - o ficheiro que constituirá o output da ferramenta, contendo as frases do ficheiro *testFIle*, com os arcos de dependência calculados e respectivas etiquetas gramaticais, em formato CoNLL 2006.

Esta ferramenta, MST Parser, também possui um avaliador. Para o correr, basta executar:

```
eval gold-file:testFile output-file:outputFile
```

onde:

- **eval** - flag que indica à ferramenta, o início do processo de avaliação.
- **gold-file:testFile** - o ficheiro/corpus dourado que será levado como base de comparação. Deverá estar em formato CoNLL 2006.
- **output-file:outputFile** - o ficheiro que será comparado com o ficheiro dourado (*testFile*). Deverá estar no mesmo formato que o *testFile*.

O resultado desta avaliação produzirá um "score" para as métricas LAS e UAS.

2.6.2 Avaliação

Aplicando um teste de "Ten Fold Cross Validation", utilizando o corpus de dependências do NLX, obteve-se os seguintes resultados, utilizando o MST Parser com o algoritmo de parsing projectivo:

Partição	LAS	LA	UAS
1	0,8940	0,9120	0,9570
2	0,8770	0,9010	0,9270
3	0,8610	0,8900	0,9070
4	0.8600	0,8870	0,9190
5	0,8790	0,9050	0,9220
6	0,8850	0,9100	0,9250
7	0,8910	0,9150	0,9360
8	0,8790	0,9000	0,9290
9	0,8910	0,9120	0,9340
10	0,8410	0,8730	0,8950
Média	**0,8758**	**0,9005**	**0,9251**

Com a execução do treino, usando o algoritmo de parsing não-projectivo, os resultados obtidos foram os seguintes:

Partição	LAS	LA	UAS
1	0,9010	0,9190	0,9570
2	0,8870	0,9060	0,9320
3	0,8730	0,9000	0,9100
4	0.8540	0,8810	0,9050
5	0,8670	0,9010	0,9130
6	0,8820	0,9090	0,9180
7	0,8800	0,9090	0,9220
8	0,8830	0,9160	0,9260
9	0,8940	0,9130	0,9370
10	0,8500	0,8820	0,8990
Média	**0,8771**	**0,9036**	**0,9219**

Como nota, é necessário referir que se modificou o número de iterações, na fase de treino, tanto para o algoritmo de parsing não-projectivo Chu–Liu/Edmonds's e o algoritmo de parsing projectivo, Eisner, sem que fossem verificadas melhorias nos "scores" finais obtidos, acima escritos.

2.7 Malt Parser

O Malt Parser [9, **?**, 13, 17, 14, 5, 15] foi desenvolvido por Joakim Nivre, Johan Hall e Jens Nilsson da Universidade de Engenharia de Sistemas e Matemática, em Vaxjo, na Suécia. Este parser é uma ferramenta orientada a dados para parsing de dependências. Malt Parser consegue integrar vários algoritmos de aprendizagem e de parsing.

Malt Parser surge como uma implementação do parsing de dependências indutivo [17], onde cada análise sintáctica de uma frase equivale à derivação de uma estrutura de dependência, e onde a aprendizagem indutiva de máquina é aplicada para guiar o parser em acções determinísticas.

Considerando uma dada frase x e a sua representação y (como sendo um grafo de dependências bem formado), uma abordagem indutiva para parsing de linguagem natural, pode ser em geral, definida em três componentes essenciais:

- Um modelo M, definindo representações para frases dadas (modelo esse constituído, por exemplo, por estruturas de dependências bem formadas).

- Um modelo estocástico parametrizado M_Θ, definindo um resultado (score) $S(x, y)$ para cada frase x e uma representação y.

- Um algoritmo de aprendizagem indutivo L para estimar os parâmetros Θ a partir de uma amostra $T_t = (x_1 : y_1, \ldots, x_n : y_n)$ de frases com as suas correctas representações (normalmente, proveniente de um corpus dourado).

O parsing de dependências indutivo é compatível com uma variedade de diferentes modelos. No entanto, este Malt Parser utiliza modelos baseados em histórico. Um tipo de modelo que pode ser definido em três passos:

- Definição de um mapeamento (um para um) entre representações sintácticas y e uma sequência de decisões $D = (d_1, ...d_m)$, tal que D determina unicamente y.

- Definição do resultado (score) $S(x, y)$ para cada decisão d_i na sequência $D = (d_1, ...d_m)$, condicionado pelo histórico $H = (d_1, ...d_{i-1})$.

- Definição de uma função Φ que agrupa pedaços do histórico em classes de equivalência, reduzindo o número de parâmetros em Θ.

Num modelo baseado em histórico, o resultado $S(x, y)$ definido pelo modelo, representa a probabilidade condicional $P(y|x)$ de analisar y, dada uma frase *x*, o que significa que a frase de input é uma variável condicionante para cada decisão na sequência de decisões:

$$P(y|x) = P(d_1, \ldots, d_m|x) = \prod_{i=1}^{m} P(d_i|\Phi(d1, \ldots, d_{i-1})) \tag{2.11}$$

Dado um modelo baseado em histórico, para uma dada frase x existem várias análises y_z. Uma vez que este parser faz uso de uma estratégia determinística, aquilo que é efectuado é a escolha de uma decisão mais provável, para uma dada análise y_j, dentro de uma sequência de decisões, $P(y_j|x)$ (sem que as outras sequências de decisões sejam verificadas) baseada na seguinte aproximação:

$$\max_j P(y_j|x) \approx \prod_{i=1}^{m} \max_i P(d_i|\Phi(d_1, \ldots, d_{i-1})) \tag{2.12}$$

Este tipo de parsing determinístico é efectuado, de modo ganancioso, procurando localmente (dentro de uma só sequência de decisões) pela decisão mais provável (óptima), na "esperança" que tal procedimento leve à uma escolha de uma decisão óptima.
Esta estratégia gananciosa melhora a eficiência de parsing, pois não existe uma procura exaustiva, no espaço (global) das sequências de decisões, pela decisão mais provável.

Com um método de aprendizagem discriminativo, é possível reduzir a aprendizagem numa questão de simples classificação, em que uma instância de input é um "histórico" parametrizado $\Phi(H, x)$, e uma classe de output é a decisão d. Usando um método supervisionado de aprendizagem, o resultado desta aprendizagem será um classificador C dado um conjunto de instâncias de treino D_t derivadas de uma amostra de uma dado corpus:

$$D_t = (\Phi(H, x), d)|O(H, x) = d, x \in T_t \tag{2.13}$$

em que Θ é uma função oracle que prevê a correcta decisão, dado o corpus dourado.

De modo a constituir uma instância específica de parsing de dependências indutivo, são necessários três passos específicos:

- Um algoritmo de parsing determinístico para construir grafos de dependências, em que um conjunto de decisões permissivas $\mathcal{D}$ e uma função oracle Θ que determina a correcta decisão dado um histórico H, são definidos. É requerido que todo o histórico possa ser caracterizado por: uma pilha σ de tokens (palavras/símbolos) parcialmente processadas, uma lista ω dos tokens restantes (dados como input), e grafo de dependências G parcialmente construído.

- Uma função de parametrização Φ usada para definir classes de equivalência sobre o histórico, e para definir vectores de características sobre as frases, tal que

$\Phi_{(1,p)} = (\Phi_1, \ldots, \Phi_p)$, onde cada característica (da palavra/token) Φ_i é uma função que mapeia uma token para a sua etiqueta morfo-sintáctica, forma lexical ou etiqueta de dependência (etiqueta de dependência inserida no grafo de dependência parcialmente construído). $\Phi_{(1,p)}$ é um modelo de características.

- aprendizagem de automática discriminativa para mapeamento do histórico relacionado com as acções tomadas pelo parser.

A arquitectura do sistema do Malt Parser foi concebida com dois objectivos em mente. O primeiro objectivo foi o de possibilitar uma flexibilização na escolha do algoritmo de parsing, modelos de características de palavras, e algoritmos de aprendizagem, a utilizar. O segundo objectivo foi o de reutilizar os componentes que são analisados na fase de aprendizagem e de parsing, por outras palavras, desenvolveu-se um sistema à semelhança de outros parsers (MST Parser, Desr Parser, por exemplo), em que um modelo é criado, na fase de treino/aprendizagem da ferramenta, e depois esse mesmo modelo é utilizado para calcular os arcos de dependências e respectivas etiquetas semânticas de um dado input. Assim sendo, a arquitectura deste parser pode ser sumarizada em três componentes:

- **Parser** - é responsável pela derivação de um grafo de dependências G, usando um algoritmo de parsing determinístico, que pode ser dividido numa série de decisões D. A fase de aprendizagem requer que cada algoritmo de parsing forneça uma implementação da função oracle Θ, usado para aprender uma sequência de decisões correctas, para um dado input. Durante a fase de parsing, o Parser apenas implementa as decisões fornecidas pelo "aprendedor" (Guide). De referir que é possível escolher algoritmos de parsing determinísticos (Nivre), ou algoritmos de parsing incrementais (Covington).

- **Guide** - O Guide tem a responsabilidade de construir um conjunto de instâncias de treino D_t para o aprendedor (Learner) durante a fase de aprendizagem, e para passar as previsões do Learner para o Parser, durante a fase de parsing. Durante a aprendizagem, o Guide cria uma instância de treino ($\Phi(H, x), d$) para cada decisão d fornecida pelo Parser, onde (Φ(H,x)) é o vector corrente de valores de características (dados a função de parametrização Φ e o estado corrente do sistema) que passa este mesmo vector para o Learner. No tempo/fase de parsing, o trainer cria um vector de características $\Phi(H, x)$ para cada pedido do Parser, passando esse mesmo vector ao Learner. A decisão prevista d é passada do Learner para o Parser. Este mecanismo/procedimento permite que o modelo de características é completamente separado do Parser, e o Learner apenas aprende tem mapeamento dos vectores de características para as decisões, sem ter saber como as características são extraídas ou que finalidade têm as decisões tomadas pelo Parser.

  A extracção das características das palavras usa a função de parametrização Φ, que

é definida por um vector de características $\Phi_{(1,p)}$, onde cada característica ϕ_i, é uma função definida por duas simples funções:

  - Função de adereço a_{ϕ_i} - identifica um token específico numa dada configuração do Parser.
  - Função de atribuição af_{ϕ_i} - captura um atributo especifico de um token:
    * Para cada i, i≥0, $\sigma[i]$ e $\omega[i]$ são funções de adereço que identificam o i + iésimo token no topo da pilha σ e identificando também o iésimo token da lista de input ω.
    * Se α é uma função de adereço, então h(α), lc(α) e rc(α) são funções de adereço, identificando o núcleo da dependência (h), a palavra filho mais à esquerda (lc) e a palavra filho mais à direita (rc), respectivamente, da token identificada por α
    * Se α é uma função de adereço, então p(α), w(α) e d(α) são funções de características, identificando as etiquetas morfo-sintácticas (p), forma da palavra (w) e o tipo de dependência (d) da token identificada por α. As funções p, w e d são chamadas de funções de atributos.

- **Learner** - tem a responsabilidade de induzir um modelo a partir do conjunto de instâncias D_t, criado pelo Guide, durante a fase de aprendizagem, e é também responsável por usar este mesmo modelo criado para prever acções do parser, durante a fase de parsing.

Quanto aos algoritmos de parsing, o Malt Parser disponibiliza três tipos de algoritmos de parsing:

- **Algoritmo de Nivre** [14] - é um algoritmo de tempo linear limitado ao parsing de estruturas projectivas. Pode ser corrido no modo arc-eager ou no modo arc-standard [13].

- **Algoritmo de Covington** [5] - é um algoritmo de tempo quadrático para estruturas de dependências sem restrições, que tenta ligar cada novo token a cada token precedente. Pode ser corrido em modo projectivo, onde a operação de ligação de tokens é restrita a estruturas de dependências projectivas. Também pode ser corrido no modo não projectivo, permitindo estruturas não projectivas (mas acíclicas).

- **Stack** [15] - é um algoritmo similar ao algoritmo de Nivre, pois também usa uma pilha e uma lista, mas diferem na atribuição de arcos de dependências, pois este algoritmo Stack, adiciona/cria os arcos de dependência entre os dois tokens das duas primeiras posições da pilha, invés de criar o arco entre o primeiro token da pilha, e o primeiro token da lista.

O algoritmo "projective stack" usa na sua essencial, as mesma transições que o "arc-standard" estando também, limitado a estruturas de dependências projectivas. Os algoritmos stack Eager e stack Lazy utilizam um passo de transição chamado Swap. Este passo de transição faz com que seja possível o que torna possível derivar árvores de dependência arbitrárias, não-projectivas. Ao passo que o stack Eager aplica este passo de transição o mais cedo possível, o algoritmo stack Lazy adia este passo para o mais tarde possível.

2.7.1 Execução

Para se realizar um treino com este parser, será necessário executar a seguinte linha

```
malt.jar -c model -i trainFile -m learn -a stacklazy
```

em que:

- **-c model** - o nome para o modelo que será criado, como resultado do treino.
- **-i trainFile** - o ficheiro de input a ser fornecido ao Malt Parser, em formato CoNLL 2006.
- **-m learn** - indica o tipo de acção do parser: treino (**learn**) ou parsing (**parse**).
- **-a stacklazy** - o tipo de algoritmo de parsing a ser utilizado. Os tipos de algoritmos a serem utilizados podem ser:
  - **nivreeager** - Nivre arc-eager
  - **nivreeager** - Nivre arc-eager
  - **covnonproj** - Covington non-projective
  - **covproj** - Covington projective
  - **stackproj** - Stack projective
  - **stackeager** - Stack eager
  - **stacklazy** - Stack lazy

De modo a realizar o parsing com esta ferramenta, será necessário executar:

```
malt.jar -c model -i testFile -o parsedFile -m parse
```

onde:

- **-c model** - o modelo com o qual o Malt Parser se baseará para efectuar o parsing.
- **-i testFile** - o ficheiro com as frases a serem analisadas, em formato CoNLL 2006.

- **-o parsedFile** - o nome do ficheiro a ser criado, que conterá as frases com os arcos de dependências e respectivas etiquetas gramaticais, calculados. Este ficheiro, vem em formato CoNLL 2006.

- **-m parse** - indica o tipo de acção do parser: treino (**learn**) ou parsing **parse**).

2.7.2 Avaliação

Como não podia deixar de ser, foi executado um treino "Ten Fold Cross Validation", utilizando o corpus de dependências do NLX, para os seguintes algoritmos:

- **Covington**

Partição	LAS	LA	UAS
1	0,8610	0,8800	0,9040
2	0,8510	0,8660	0,8860
3	0,8370	0,8650	0,8720
4	0,8190	0,8420	0,8620
5	0,8500	0,8670	0,8890
6	0,8250	0,8490	0,8640
7	0,8580	0,8750	0,8850
8	0,8430	0,8670	0,8740
9	0,8750	0,8870	0,9010
10	0,8220	0,8400	0,8650
Média	**0,8441**	**0,8638**	**0,8802**

- **Covington non projective**

Partição	LAS	LA	UAS
1	0,8940	0,9220	0,9330
2	0,8800	0,9050	0,9150
3	0,8720	0,9060	0,9100
4	0,8370	0,8790	0,8760
5	0,8750	0,9150	0,9060
6	0,8640	0,8940	0,9000
7	0,8950	0,9200	0,9210
8	0,8850	0,9100	0,9230
9	0,9070	0,9250	0,9350
10	0,8420	0,8790	0,8890
Média	**0,8751**	**0,9055**	**0,9108**

- **Nivre eager**

Partição	LAS	LA	UAS
1	0,8890	0,9120	0,9320
2	0,8730	0,9000	0,9030
3	0,8480	0,8800	0,8840
4	0,8240	0,8610	0,8640
5	0,8690	0,8890	0,9070
6	0,8490	0,8770	0,8850
7	0,8800	0,9060	0,9090
8	0,8600	0,8930	0,8960
9	0,8930	0,9090	0,9250
10	0,8370	0,8680	0,8800
Média	**0,8622**	**0,8895**	**0,8985**

- **Nivre arc standard**

Partição	LAS	LA	UAS
1	0,8460	0,8670	0,8860
2	0,8230	0,8450	0,8600
3	0,8290	0,8560	0,8670
4	0,8230	0,8390	0,8630
5	0,8360	0,8450	0,8720
6	0,8200	0,8510	0,8560
7	0,8500	0,8710	0,8800
8	0,8180	0,8450	0,8550
9	0,8730	0,8880	0,9000
10	0,8100	0,8290	0,8520
Média	**0,8328**	**0,8545**	**0,8691**

- **Stack projective**

Partição	LAS	LA	UAS
1	0,8540	0,8190	0,8950
2	0,8440	0,8820	0,8780
3	0,8460	0,8850	0,8830
4	0,8380	0,8760	0,8800
5	0,8560	0,8940	0,8890
6	0,8290	0,8740	0,8710
7	0,8620	0,9040	0,8960
8	0,8250	0,8660	0,8610
9	0,8720	0,9030	0,8970
10	0,8220	0,8590	0,8710
Média	**0,8448**	**0,8834**	**0,8821**

- **Stack eager**

Partição	LAS	LA	UAS
1	0,8780	0,9000	0,9200
2	0,8660	0,8930	0,9050
3	0,8850	0,9100	0,9250
4	0,8480	0,8830	0,8890
5	0,8840	0,9120	0,9200
6	0,8560	0,8920	0,8940
7	0,8980	0,9250	0,9270
8	0,8510	0,8900	0,8930
9	0,8940	0,9220	0,9250
10	0,8530	0,8850	0,9020
Média	**0,8713**	**0,9012**	**0,9100**

- **Stack lazy**

Partição	LAS	LA	UAS
1	0,8740	0,9000	0,9150
2	0,8700	0,8980	0,9110
3	0,8820	0,9140	0,9210
4	0,8560	0,8940	0,8970
5	0,8910	0,9150	0,9300
6	0,8610	0,9000	0,9000
7	0,8900	0,9180	0,9280
8	0,8720	0,8990	0,9120
9	0,9080	0,9260	0,9340
10	0,8630	0,8900	0,9120
Média	**0,8767**	**0,9054**	**0,9160**

É importante referir que para estes resultados obtidos, foi usado um algoritmo de aprendizagem que se baseia no algoritmo *Support Vector Machines*. Foi usado outro algoritmo de aprendizagem baseado no algoritmo *Large Linear*, mas apenas se verificou um decréscimo nos resultados.

2.8 Considerações Finais

Neste capítulo, apresentou-se um estudo sobre parsers de dependências. Foram apresentados cinco parsers de dependências: ISBN Parser, DeSR Parser, KSDEP Parser, MST Parser e Malt Parser, sendo que, pelos resultados apresentados, o MST Parser foi o analisador de dependências escolhido. Este parser de dependências será utilizado para concretizar as ferramentas *LX Dep Parser* e *Marcador Semântico*.

De modo a justificar a escolha do parser de dependências, observemos o seguinte Quadro:

Parser	LAS	LA	UAS
MST (proj)	0,8758	0,9005	**0,9251**
Malt	0,8767	**0,9054**	0,9160
MST (não-proj)	**0,8771**	0,9036	0,9219

O MST Parser é a ferramenta que melhor resultados produz, após um teste de "Ten Fold Cross Validation". As duas ferramentas Matl Parser e MST Parser, foram os analisadores de dependências que melhor se distinguiram entre os restantes.

Na Tabela anterior é possível observar que o Malt Parser consegue bater por uma mínima margem o MST Parser, na métrica LA, mas o parser MST correndo em modo "projectivo" e modo "não projectivo", é possível constatar que consegue bater o Malt Parser, nas métricas UAS e LAS.

Posto este cenário, a ferramenta de parser de dependências escolhida é o MST Parser. Uma vez que o corpus de dependências do NLX utilizado, contendo apenas as etiquetas gramaticais, foi um corpus em que o seu desenvolvimento/melhoramento estava em decurso e que também este corpus não possuía mais do que mil duzentas e quatro frases, é necessário referir que os resultados obtidos são excelentes.
Observando o seguinte Quadro, os únicos parsers testados com um corpus de dependências em português:

Parser	LAS	UAS
MST (não-proj)	0,8680	0,9140
DeSR	0,8080	0,8680

não conseguem superar os resultados obtidos pelos parsers MST e Malt, com o corpus de dependências do NLX, o que é excelente, pois significa que estamos perante um corpus de dependências consistente, que tem a capacidade de levar certas ferramentas de parsing de dependências a produzir resultados muito acima da média, embora neste caso, os resultados cheguem a ser mesmo mais elevados que outros resultados verificados.

No próximo Capítulo, será descrita a ferramenta Web *LX Dep Parser* e no Capítulo 4, será descrita a ferramenta *Marcador Semântico*.

Capítulo 3

LX Parser de Dependências

3.1 Introdução

Escolhido o **M**aximum **S**panning **T**ree Parser para prosseguir com o trabalho, decidi deixar o meu contributo no repositório de ferramentas Web do Grupo NLX, implementando a ferramenta Web *LX Dep Parser*[1] (LX Parser de Dependências).

A ferramenta *LX Dep Parser* tem uma interface simples (Figura 3.1) que recebe uma frase de texto (escrita em Língua Portuguesa) do utilizador, e devolve a estrutura de dependências da frase inserida, com os arcos de dependências e respectivas etiquetas gramaticais (apenas).
Como é possível ver na Figura 3.1 após a inserção da frase de exemplo "A Maria tem razão", clicando no botão "Analisar", obtém-se a imagem do grafo de dependências da frase, onde podemos observar as seguintes ralações gramaticais das palavras na frase:

- Arco de dependência de "tem" para "Maria", com a etiqueta gramatical "SJ" (*Subject*).
- Arco de dependência de "tem" para "razão", com a etiqueta gramatical "DO" (*Direct Object*).
- Arco de dependência de "tem" para ".", com a etiqueta gramatical "PUNCT" (*Punctuation*).
- Arco de dependência de "tem" para "Root", com a etiqueta gramatical "ROOT", que indica que a palavra "tem" é a raiz da frase.
- Arco de dependência de "Maria" para "a", com a etiqueta gramatical "SP" (*Specifier*).

Passemos então a explicar o funcionamento da ferramenta *LX Dep Parser*.

De modo a explicar o funcionamento do *LX Dep Parser*, comecemos por explicar o funcionamento do MST parser:

[1] http://lxcenter/services/en/LXServicesParserDep.html

Figura 3.1: A interface da ferramenta Web *LX Dep Parser*

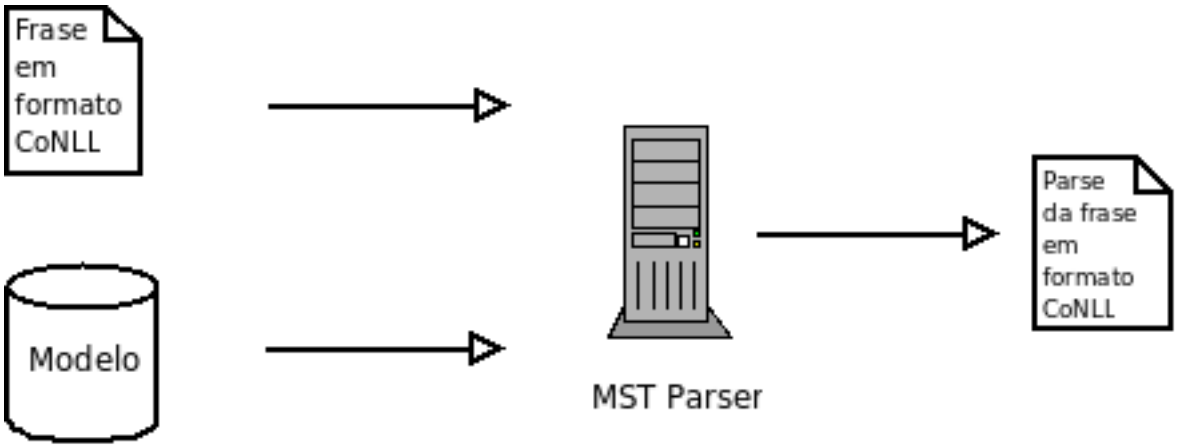


Figura 3.2: Diagrama do funcionamento do MST Parser

3.2 Funcionamento

A ferramenta *LX Dep Parser* utiliza o MST Parser para calcular os arcos de dependências e respectivas etiquetas gramaticais. Como ilustrado na Figura 3.2, o funcionamento do parser de dependências MST consiste em receber como input um ficheiro de texto com uma ou mais frases em formato CoNLL 2006 e também o modelo resultante do treino com um corpus de dependências. O MST Parser procede então ao carregamento do modelo, fazendo de seguida o parsing das frases dadas como input, produzindo um ficheiro com as frases no mesmo formato CoNLL 2006, já com os arcos de dependências e respectivas etiquetas, calculados.

Uma vez que o MST Parser é um "módulo" da ferramenta *LX Dep Parser*, será necessário modificar o input e o output do MST Parser.

Na Figura 3.3 podemos observar o diagrama que mostra o funcionamento do *LX Dep Parser*. Ao *servidor local* é passada a frase que o utilizador digitou, através de uma aplicação *cliente local* que é lançada pelo servidor do Grupo NLX aquando de recepção de um pedido do utilizador.
A frase é então processada e passada a um *Pipeline* de ferramentas, com a finalidade de converter a frase anotada para o formato CoNLL 2006, efectuar o parsing sobre a frase já convertida para o formato CoNLL 2006, e de desenhar o grafo de dependências da mesma frase. No final o utilizador visualizará o grafo de dependências da frase que digitou.

Esta ferramenta *LX Dep Parser* terá de constituir-se como uma ferramenta que seja capaz de receber como input uma frase em texto limpo. Terá também que ser eficiente, para que a ferramenta Web não proceda ao carregamento do modelo do *MST Parser* sempre que seja necessário processar uma frase, pois este é um processo que demora cerca de quatro

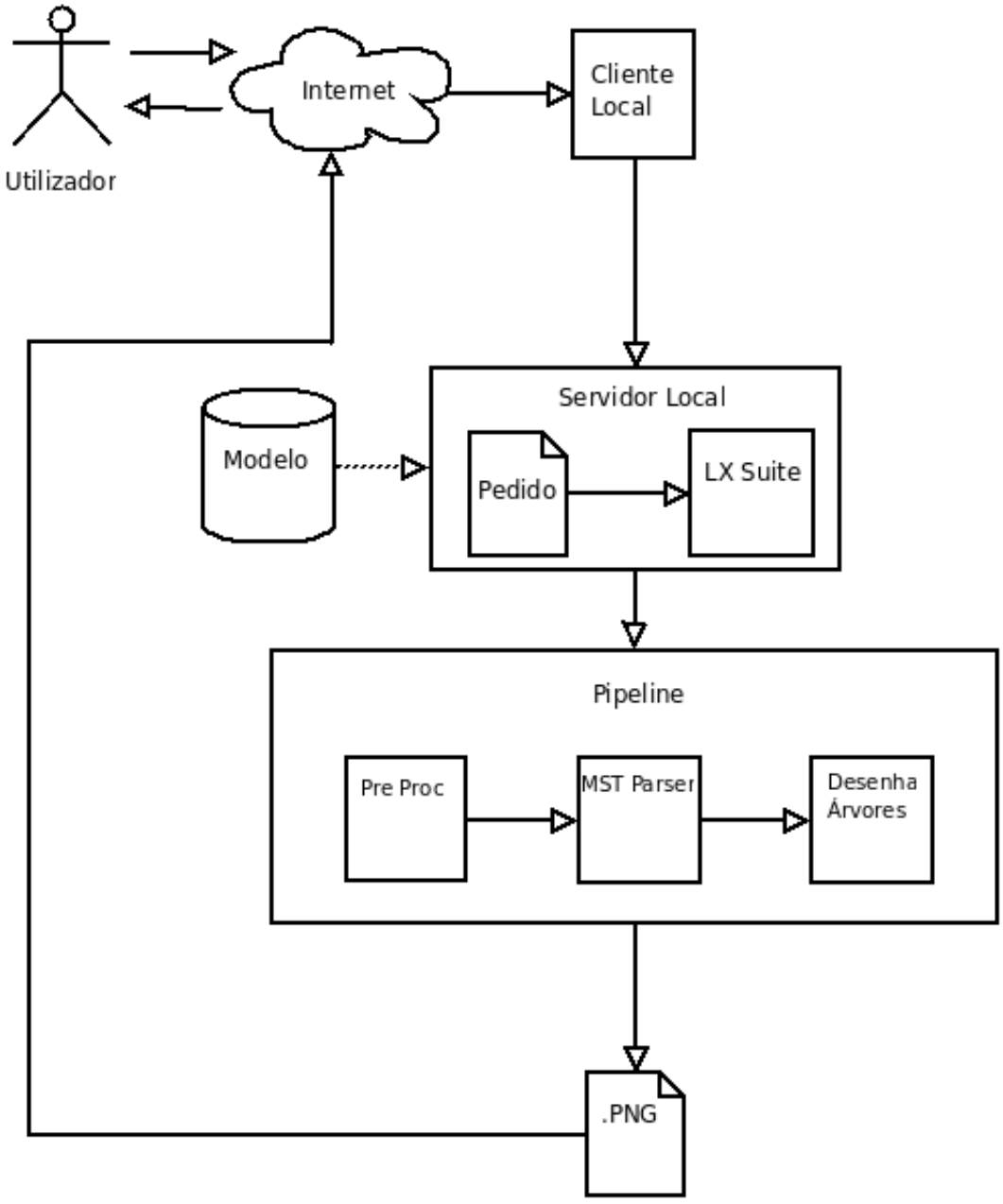


Figura 3.3: Diagrama da ferramenta Web *LX Dep Parser*

segundos.
Também será de levar em conta que esta ferramenta ficará alojada num servidor, o que por sua vez nos traz a questão da manutenção. Dado que as imagens das estruturas de dependências das frases serão geradas em tempo de execução e armazenadas no servidor, é preciso que esta ferramenta seja capaz de efectuar uma verificação periódica sobre as imagens mais antigas e que proceda à eliminação destas mesmas imagens, mais antigas.

Assim sendo, o funcionamento do *LX Dep Parser*, decorre como se descreve nas secções seguintes.

3.2.1 Servidor Local

O funcionamento do *LX Dep Parser* começa pelo arranque do servidor local, localizado no servidor do Grupo NLX. Este servidor local, representado na Figura 3.3, começa por executar o carregamento do modelo obtido pelo MST Parser a partir do corpus de dependências com as etiquetas gramaticais. Este procedimento que fazia parte do parser, foi transferido para o servidor local que passa agora a exercer essa função, pois o objectivo desta "transferência" passa por carregar o modelo apenas uma vez, realizando o parsing das frases pedidas pelos vários utilizadores. Foi com este objectivo que se teve em conta a criação deste servidor local, pois era necessário organizar numa aplicação o carregamento do modelo e ficar à escuta por pedidos (frases em texto limpo) do cliente local, para processar.

Do lado do utilizador, é submetido um pedido que é comunicado ao servidor do Grupo NLX. O servidor NLX lança então a aplicação *cliente local* que irá comunicar com o *servidor local*, através de um porto designado. Esta aplicação transmite ao servidor local a frase em texto limpo, a ser processada. Aquando da recepção por parte do servidor local, é gerado um ficheiro onde será guardado a imagem final com os arcos de dependência calculados. A geração desse ficheiro consiste na criação de um ficheiro com extensão .png, cujo nome conterá um inteiro que representa o tempo em milissegundos desde 1 de Janeiro de 1977 até à data da criação do ficheiro.
De modo a garantir que não ocorram problemas caso a geração de dois ou mais ficheiros aconteça ao mesmo tempo, são gerados ficheiros temporários, o que garante que na altura da criação do ficheiro, esse ficheiro contenha um nome único, ao qual depois é concatenado o inteiro contento o tempo em milissegundos.
A concatenação do tempo passado em milissegundos ao nome do ficheiro é efectuado pois o servidor local efectua periodicamente um "inspecção" dos ficheiros .png, eliminando todos os ficheiros com uma determinada duração de vida.

O servidor local é também responsável por executar, sobre a frase submetida pelo cliente, um conjunto de aplicações chamado de LX Suite, que vai agregar informação sintáctica a

cada token da frase.

LX-Suite

O LX Suite é uma ferramenta que está disponibilizada on-line, sendo composta por uma série de outras ferramentas activadas numa determinada ordem:

- **LX-Chunker** - Tem a função de marcar os limites de uma frase com "<s>...</s> ", e de marcar também os parágrafos com " ... "

- **LX-Tokenizer** - Tem a função de segmentar o texto em "símbolos" usando o espaço em branco como separador, como podemos ver no seguinte exemplo (de notar que nos exemplos, em vez de espaços em branco, utilizar-se-á o símbolo "|", para explicitar a separação):

  um exemplo → |um|exemplo|

  Esta ferramenta também consegue efectuar a expansão de contracções:

  do → |de_|o|

  O LX-Tokenizer consegue tratar de ambiguidades. Por exemplo, a palavra "deste" (dependendo da sua ocorrência) pode ser "tokenizada" de duas maneiras:

  deste → |deste|
  quando se trata de um verbo
  deste → |de_|este|
  quando ocorre como uma contracção

- **LX-Tagger** - Esta ferramenta atribui uma etiqueta morfo-sintáctica a cada símbolo (token):

  o/SP João/PNM

- **LX-Featurizer** - Atribui ás palavras, valores de flexão referentes às categorias nominais, tais como: Género (**m**asculino ou **f**eminino), Número (**s**ingular ou **f**eminino), e se aplicável, Pessoa (**1**ª, **2**ª ou **3**ª):

  os/DA gatos/CN → os/DA#mp gatos/CN#mp

  Também possui a capacidade de atribuir flexão de grau (**dim**inutivo, **sup**erlativo e **comp**arativo):

os/DA gatinhos/CN → os/DA#mp gatinhos/CN#mp-dim

- **LX-Lemmatizer** - A função de esta ferramenta é a de associar um lema à palavra a partir das categorias nominais (adjectivos, nomes comuns e particípios passados). O lema corresponde à forma da palavra que se encontrará no dicionário, tipicamente na forma do singular, masculino. Esta informação é concatenada ao símbolo com o delimitador "/":

  gatas/CN#fp → gatas/GATO/CN#fp

- **LX-Lemmatizer and Featurizer (verbo)** - Tem a mesma função que o LX-Lemmatizer, mas aplicado aos verbos, sendo que o lema corresponderá à forma infinitiva do verbo:

  escrevi/V → escrevi/ESCREVER/V#ppi-1s

Tomando como exemplo a seguinte frase:

A Maria tem razão.

o resultado que sai para o módulo *MST Parser* será:

```
<p> <s> A/DA#fs Maria/PNM tem/TER/V#pi-3s
    razão/RAZÃO/CN#gs ./PNT </s> </p>
```

3.2.2 Pipeline

O *Pipeline* de ferramentas (no *LX Dep Parser*) recebe a frase devidamente anotada, com a finalidade de converter a anotação da frase para o formato CoNLL 2006 (*Pre Proc*), calcular os arcos de dependências e respectivas etiquetas gramaticais (*MST Parser*), e efectuar a representação desses mesmos arcos (grafo de dependências) produzindo uma imagem em formato .png (*Desenha Árvores*), que será visualizada pelo utilizador.

Pre Proc

De modo a que o *MST Parser* consiga efectuar a tarefa de parsing sobre a anotação da frase recebida, é necessário converter essa anotação para o formato CoNLL 2006, usando a ferramenta *Pre Proc* (Pré-Processamento).
Com anotação da frase "A Maria tem razão":

```
<p> <s> A/DA#fs Maria/PNM tem/TER/V#pi-3s
    razão/RAZÃO/CN#gs ./PNT </s> </p>
```

a ferramenta *Pre Proc* coloca a anotação da frase no formato CoNLL 2006:

1	A	_	DA	DA	fs	0	_	_	_
2	Maria	_	PNM	PNM	_	0	_	_	_
3	tem	TER	V	V	_	0	_	_	_
4	razão	RAZÃO	CN	CN	gs	0	_	_	_
5	.	_	PNT	PNT	_	0	_	_	_

Será esta representação da frase em formato CoNLL 2006 a ser analisada pelo *MST Parser*.

MST Parser

É com base no resultado obtido pela ferramenta *Pre Proc* que o cálculo dos arcos de dependências será efectuado. De referir que a coluna da cabeça da dependência, que indica o posição da palavra do qual a palavra corrente depende, está a 0 pois este é um valor por omissão necessário, para o parser prosseguir com as suas tarefas. Após a análise da frase pela ferramenta *Parser*, este é será o resultado:

1	A	_	DA	DA	fs	2	SP	_	_
2	Maria	_	PNM	PNM	_	3	SJ	_	_
3	tem	TER	V	V	_	0	ROOT	_	_
4	razão	RAZÃO	CN	CN	gs	3	DO	_	_
5	.	_	PNT	PNT	_	3	PUNCT	_	_

onde é possível constatar que as colunas sete e oito que dizem respeito respectivamente à cabeça da dependência (arco de dependência) e à etiqueta gramatical que define essa mesma dependência, estão agora preenchidas.

Com os arcos de dependências e respectivas etiquetas gramaticais calculados, a ferramenta *Desenha Árvores* desenhará o grafo de dependências, a ser apresentado ao utilizador.

Desenha Árvores

Após a tarefa de parsing, é com base no resultado da ferramenta *Parser* que se desenhará a árvore de dependências. Esta tarefa ficará a cargo da ferramenta *DrawTrees*, que a partir da frase em formato CoNLL 2006 já com as dependências calculadas e com base numa API de JAVA chamada "What's Wrong With My NLP?", [2] é efectuada a imagem da frase inserida pelo utilizador (Figura 3.1). Para além dos arcos de dependências e suas etiquetas gramaticais, também mostra a informação referente a cada palavra, tal como: a sua posição na frase, o lema da palavra (se disponível), a etiqueta morfo-sintáctica, valores de flexão da palavra (se disponível).

Finalizando, depois de ser gravada a imagem da árvore no ficheiro, o utilizador visualiza a imagem no seu browser.

[2] http://code.google.com/p/whatswrong/

3.3 Considerações Finais

Este capítulo explicou o funcionamento da ferramenta *LX Dep Parser*, assim como os módulos que integram esta ferramenta Web.
No próximo capítulo, será descrita a ferramenta *Marcador Semântico* que visa efectuar o markup semântico das páginas Web, escritas em Português.

Capítulo 4

Marcador Semântico

4.1 Introdução

Marcador Semântico é o nome da ferramenta que visa efectuar a marcação semântica das páginas web escritas em Português.

Tal como o *LX Dep Parser*, esta ferramenta utiliza o MST Parser para calcular os arcos de dependências e respectivas etiquetas gramaticais e semânticas. Para além de um *Pipeline* (contendo as ferramentas *Pre Proc* e *MST Parser*) que faz parte do *Marcador Semântico*, existe também uma ferramenta chamada de *Etiquetador* (constituído pelas ferramentas *Analisador de Frases* e *RDF/XML Writer*) que tem a função de retirar a relação semântica entre as palavras, e registá-las sobre a forma de triplos na linguagem RDF/XML.
Exemplificando com a frase "A Maria tem razão.", o *Pipeline* produz o seguinte grafo de dependências:

O *Analisadore de Frases* (*Etiquetador*) obtém os seguintes triplos através do grafo de dependências:

(Root, ROOT, tem)
(tem, ARG1, Maria)
(tem, ARG2, razão)

E o *RDF/XML Writer* (*Etiquetador*) escreve a representação semântica da frase em linguagem de marcação RDF/XML:

```
<?xml version="1.0"?>

<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
```

```
        xmlns:ARG1="Argument_1/"
        xmlns:ARG2="Argument_2/"
        xmlns:ROOT="Root/"
        xmlns:Value="Word_Value/"
>

    <rdf:Description rdf:about="Sentence_0-Word_3"
        Value:Word="tem">

        <ARG1:Argument_1 rdf:resource="Sentence_0-Word_2"
            Value:Word="Maria"/>

        <ARG2:Argument_2 rdf:resource="Sentence_0-Word_4"
            Value:Word="razão"/>
    </rdf:Description>

    <rdf:Description rdf:about="Sentence_0" >
         <ROOT:Root rdf:resource="Sentence_0-Word_3"
            Value:Word="tem"/>
    </rdf:Description>

</rdf:RDF>
```

Logo que o corpus de dependências do Grupo NLX anotado com as etiquetas semânticas ficou disponível, o mesmo foi utilizado para treinar o *MST Parser* de modo a produzir um modelo que seria utilizado para calcular os arcos de dependências e respectivas etiquetas gramaticais e semânticas (grafo de dependências), para uma dada frase.

Para se observar o resultado final da ferramenta *Marcador Semântico*, segue no Apêndice B a representação semântica de algumas frases retiradas do corpus de dependências do Grupo NLX. No Apêndice A seguem os grafos de dependências e respectivos triplos das frases cujas representação semânticas se encontram no Apêndice B.

Nas próximas subsecções dá-se uma breve explicação sobre a linguagem de marcação RDF/XML com a qual se representa a semântica de um texto (frase a frase), e também sobre o corpus usado para treinar o parser de dependências *MST Parser*, usado pela ferramenta *Marcador Semântico*.
São também abordadas neste capítulo, as ferramentas: *LX-Suite*, *Pre Proc* e *MST Parser* (*PipeLine*), *Analisador de Frases* e *RDF/XML Writer* (*Etiquetador*).

4.1.1 Resource Description Framework

A linguagem de marcação RDF/XML insere-se na Web Semântica como uma especificação formal para atribuir uma descrição sobre conceitos, termos e relações de um dado domínio.

A Web Semântica é um termo definido pela WC3[1] que descreve métodos e terminologias que permitem que as máquinas possam compreender a semântica da informação presente na Web. Tecnologias como OWL (Ontology Web Language) e RDF (Resource Description Framework), podem ser usadas para descrever a semântica de uma dada informação, por outras palavras, é possível representar entidades arbitrárias tais como: pessoas, peças de automóveis, etc .
Com estas linguagens de marcação é possível descrever a informação de modo a que possam ser processado por, por exemplo, agentes automáticos.

Assim sendo, a escolha da linguagem de marcação para representar/descrever os triplos obtidos com a ferramenta *Analisador de Frases*, recai sobre a linguagem RDF/XML uma vez que permite a representação de informação através de relações ternárias constituídas por um sujeito, um predicado e por um objecto (triplo), que vai ao encontro do que pretendemos representar.

A linguagem XML, só por si, embora seja utilizada para descrever sintacticamente os elementos presentes num discurso/texto, não será a linguagem apropriada pois não iremos querer descrever a informação/valor que as palavras representam, mas sim as relações semânticas entre essas mesmas palavras de uma dada frase.

Uma ontologia é um modelo de dados que representa um conjunto de conceitos dentro de um domínio e os relacionamentos entre estes. A linguagem OWL é constituída sobre a linguagem RDF, e uma vez que não iremos definir conjuntos ou tipos de objectos (classes), mas sim o relacionamento entre objectos, a linguagem RDF/XML será a mais apropriada.

RDF/XML é tecnologia proposta pela W3C para codificar conhecimento de modo a ser processado por aplicações. RDF/XML usa linguagem RDF concebida para expressar proposições, com um vocabulário específico, encapsulada em linguagem XML.

RDF (por si só) é também uma linguagem que usa metadados para representar sobre recursos na World Wide Web, tais como: título, autor, direitos de autor, etc.

A linguagem RDF é definida por uma sintaxe abstracta. Com essa sintaxe, é possível constituir um grafo contendo nós e arcos direccionados, que emparelham pares de nós, configurando triplos RDF. Cada triplo contém um sujeito, predicado e objecto, como se pode verificar na Figura 4.1.

Um grafo RDF é um conjunto de triplos RDF, sendo que um grafo RDF é constituído por um conjunto de nós (Nós esses que definem o sujeito e o objecto dos triplos que constituem o grafo RDF).

[1] W3C é um consorcio criado em 1994, que entre outras coisa, tem a tarefa de definir ferramentas padrão para o desenvolvimento da Web

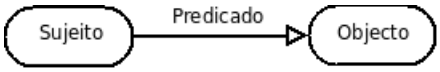


Figura 4.1: Um exemplo de um triplo em RDF que convencionalmente é definido pela ordem: sujeito, predicado, objecto.

Como tal, um triplo RDF contém três componentes:

- Sujeito
- Objecto
- Predicado

Os três componentes de um triplo RDF (Sujeito, Predicado e Objecto) podem assumir os tipos:

- **RDF URI Reference** (Uniform Resource Identifier) - que dentro de um grafo RDF corresponde a uma string Unicode, codificada em UTF-8. De notar que duas RDF URI são iguais se e só se forem iguais, caracter a caracter, comparadas como duas string Unicode.
- **Literal** - podem ser compostos por um ou dois componentes nomeados por uma string. Um Literal pode ser:
  - **Plain literals** - compostos por uma string, e opcionalmente uma "etiqueta de linguagem". Uma etiqueta de linguagem serve, por exemplo, como uma informação para caracterizar a linguagem em que a informação está escrita ("chat@en").
  - **Typed literals** - compostos por uma string e um datatype URI, compondo um RDF URI Reference. Um datatype é usado em RDF como representação de valores, tais como: uma string, um inteiro, uma data, etc.
- **Blank Node** - Num grafo RDF, blank nodes não possuem uma RDF URI Reference. Por vezes, é necessário que um blank node num grafo RDF, seja referido em vários locais do grafo, como sujeito ou objecto de vários triplos RDF. Pode ser usado o atributo "rdf:nodeID" para diferenciar blank nodes.

De referir que os nós Sujeito apenas podem assumir os tipos **RDF URI Reference** ou **Blank Nodes**. O elemento Objecto de um triplo podem ser um nó identificado por uma **RDF URI Reference**, um (**Blank nodes** ou então pode ser um **Literals**. E os Predicados apenas podem ser do tipo **RDF URI Reference**.

Mais à frente será explicado com maior detalhe, a sintaxe da linguagem de marcação RDF/XML para usada para codificar a representação semântica de frases de um texto.

Falemos então sobre o corpus de dependências com etiquetas semânticas, do Grupo NLX, usado para produzir um modelo de treino para a ferramenta *MST Parser*.

4.1.2 Corpus

O corpus de dependências CINTIL, desenvolvido no Grupo NLX, é constituído por frases do foro jornalístico. Possui mil duzentas e quatro frases anotadas com informação linguística, tal como a etiqueta morfo-sintáctica de cada palavra, lema da palavra (caso exista), e toda a informação necessária para constituir o formato CoNLL 2006 (ver Secção 1.5)

Este corpus foi criado com o apoio de uma gramática que produz diversas análises gramaticais possíveis (parses) para uma dada frase. O anotador humano escolhe a análise correcta a partir das que foram gerados pela gramática através de decisões binárias. Por exemplo, para a frase

O João viu a Maria com os binóculos.

o anotador humano decide, em termos de representação semântica, guardar a representação que corresponde à situação em que o João utilizou os binóculos para observar a Maria, ou guardar a representação que corresponde á situação em que a Maria, contendo em sua posse um par de binóculos, foi observada pelo João.

Após esta fase de anotação do corpus, que consiste na escolha de análises para um dado conjunto de frases, essas mesmas análises passam por uma ferramenta que após o seu processamento, devolve as frases anotadas no formato CoNLL 2006. Neste ponto, as frases no formato CoNLL já possuem os arcos de dependências calculados, assim como as respectivas etiquetas gramaticais.

O corpus utilizado para a ferramenta *Marcador Semântico*, contém algumas alterações relativamente ao corpus que foi usado para treinar o MST Parser de modo a produzir um modelo a ser utilizado com a execução da ferramenta Web *LX Dep Parser*.

Essas alterações, entre outras, consistem na substituição de etiquetas gramaticais por etiquetas semânticas.

As etiquetas gramaticais tais como: SJ (Sujeito), DO (Objecto Directo) ou OBL (Complemento Obliquo), são substituídas por etiquetas semânticas. “ARGx” (Argumento) em que x representa o número do argumento que a palavra dependente constitui em relação à palavra núcleo (por norma, um verbo).

Uma vez que as etiquetas gramaticais “C” (Complemento), indicam que em relação a palavra núcleo, apenas existe um argumento, estas etiquetas gramaticais serão substituídas pela etiqueta semântica “ARG0”, como e possível ver na Figura 4.2.

As etiquetas gramaticais “M” (Modificador) foram completadas (manualmente, através

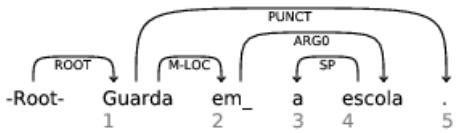


Figura 4.2: Exemplo de uma etiqueta semântica M-LOC e ARG0

de um anotador humano) por uma etiqueta de segundo nível (etiqueta semântica) que indicará com maior nível de precisão semântica, o tipo de modificador que a palavra dependente constitui à palavra núcleo.

Na Figura 4.2 é possível observar que a preposição "em_" depende da palavra "guarda", modificando em termos de localização, a palavra "Guarda". É este tipo relação entre as palavras numa frase que queremos capturar, pois a informação relativa ás relações semânticas entre as palavras numa dada frase representará a semântica (significado) de uma dada frase.

É no entanto importante referenciar o resultado para um teste do tipo "Ten Fold Cross Validation" com o corpus que contém as etiquetas semânticas:

Parser	LAS	UAS
MST (proj)	0,8533	0,9309
MST (não-proj)	0,8511	0,9279

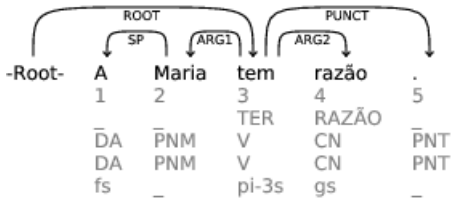
A lacuna entre os resultados obtidos com o MST Parser para as métricas *LAS* e *UAS*, é maior com este corpus de dependências com as etiquetas semânticas do que com o corpus de dependências sem as etiquetas semânticas (ver Secção 2.6.2). O que pode ajudar a explicar tal ocorrência é o facto de o corpus com etiquetas semânticas ter mais etiquetas para rotular os arcos de dependências (trinta e quatro) do que o corpus sem as etiquetas semânticas (cerca de vinte e quatro). Uma vez que tanto o corpus de dependências com as etiquetas semânticas como o corpus de dependências sem etiquetas semânticas são corpus relativamente pequenos, com mil duzentos e quatro frases, uma solução óbvia para diminuir a lacuna entre as duas métricas *LAS* e *UAS* no corpus de dependências com as etiquetas semânticas seria o aumento em termos de frases anotadas desse mesmo corpus.

Interessa referir que não existe outro corpus de dependências em língua Portuguesa com etiquetas semânticas. Como tal podemos afirmar que estes resultados obtidos com o MST Parser são os melhores obtidos até agora para o Português.

4.2 Representação Semântica

A ferramenta *Marcador Semântico* foi criada com o intuito de extrair uma representação da semântica de frases e consequentemente de um texto de input.
Numa frase, é "capturada" a representação semântica sob a forma de triplos, que são formados a partir das relações de dependências (arcos) entre duas palavras, sendo essa

representação semântica da frase codificada em de marcação linguagem RDF/XML.
Por exemplo, para a frase: "A Maria tem razão.":



teria como representação semântica, os seguintes triplos.

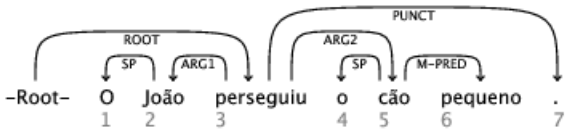
(Root, ROOT, tem)
(tem, ARG1, Maria)
(tem, ARG2, razão)

Com estes triplos, é possível encontrar os dois argumentos que possuem uma relação semântica com a raiz da frase (a palavra "tem"): a palavra "Maria" como argumento 1 e a palavra "razão" como argumento 2.
O triplo **(tem, PUNCT, .)** não fará parte do lote de triplos acima escrito, pois "não" contém informação semântica que se considere importante. No triplo **(tem, PUNCT, .)**, a pontuação "." com a etiqueta gramatical "PUNCT" (Punctuation) não contém informação relevante para a semântica da frase. De referir que para toda a pontuação de uma frase, qualquer que seja o núcleo da dependência que envolva a pontuação, a etiqueta gramatical será sempre "PUNCT".
Também o triplo **(Maria, SP, A)** não será considerado pois a representação deste triplo na linguagem de marcação RDF/XML, ultrapassa a capacidade de expressão da própria linguagem RDF/XML. Para este tipo de relação gramatical (quantificadores esses que no corpus de dependências do Grupo NLX dependem de outras palavras com a etiqueta gramatical "SP"), segundo [4] necessitamos de uma linguagem de segunda ordem de modo a ser possível expressar quantificadores sobre indivíduos/componentes (Barwise & Cooper, 1980). Uma vez que a linguagem de marcação RDF/XML não é uma linguagem de primeira ou segunda ordem, não será possível expressar com exactidão este tipo de relações gramaticais.

Advérbios, preposições e adjectivos que sejam modificadores de verbos ou nomes comuns também contêm um valor semântico e como tal, também deverão ser levados em conta para a formação de triplos. Por exemplo, consideremos a seguinte frase:

O João perseguiu o cão pequeno.

com o adjectivo "pequeno" para descrever o cão, e consideremos também o grafo de dependências da mesma frase:



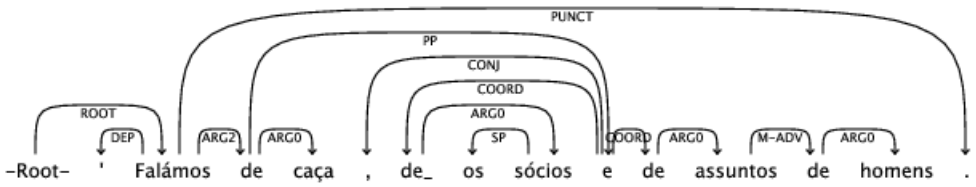


Figura 4.3: Frase com uma conjunção coordenativa

Podemos observar que os triplos gerados com esta frase seriam:

(Root,	ROOT,	perseguiu)
(perseguiu,	ARG1,	João)
(perseguiu,	ARG2,	cão)
(cão,	M-PRED,	pequeno)

onde o triplo **(cão, M-PRED, pequeno)** mostra que o adjectivo "pequeno" tem uma relação semântica de "perdicado modificador" (M-PRED) em relação à palavra "cão". Este tipo de etiquetas semânticas (M-X) serão sempre levadas em conta para a obtenção de triplos a serem escritos para o ficheiro RDF/XML.

Como já referido, algumas relações gramaticais deverão ser observadas e obtidas sobre a forma de um triplo, pois ajudarão a melhor estruturar a representação da semântica da frase no ficheiro RDF/XML. Esse tipo de relações gramaticais são as relações gramaticais que envolvam conjunções coordenativas. Conjunções são palavras invariáveis que servem para conectar orações ou dois termos de uma mesma função sintáctica, estabelecendo entre eles uma relação de coordenação. Estas conjunções coordenativas que ligam vários itens possui normalmente a conjunção "e". Este tipo conjunção coordenativa liga-se (arcos de dependências) aos vários "itens" através da etiqueta semântica/gramatical "COORD". Exemplificando com outra frase, na Figura 4.3, retirada directamente do corpus, podemos ver que os triplos para a frase serão os seguintes:

(Root,	ROOT,	Falámos)
(Falámos,	ARG2,	e)
(e,	COORD,	caça)
(e,	COORD,	sócios)
(e,	COORD,	assuntos)
(assuntos,	M-ADV,	de)
(de,	ARG0,	homens)

Uma vez que a conjunção coordenativa [**?**, p.551] "e" tem o papel sintáctico de unir constituintes do mesmo nível categorial ("de caça", "de sócios" e "de assuntos"), esta conjunção terá de ser registada como constituinte (Sujeito e Objecto) de alguns triplos. Nomeadamente o triplo *(Falámos, ARG2, e)* que liga a raiz da frase à conjunção, e os triplos *(e, COORD, caça), (e, COORD, caça), (e, COORD, caça)* que indicará a ligação da conjunção "e" aos itens que estão coordenados.

Podemos também reparar que a preposição "de" na décima posição da frase foi integrada no triplos *(assuntos, M-ADV, de)* e *(de, ARG0, homens)*, uma vez que é modificadora da palavra "assuntos".

4.3 Funcionamento

A Figura 4.4 esquematiza o funcionamento da ferramenta *Marcador Semântico*. Esta ferramenta recebe um modelo criado pelo treino[2] do MST Parser que será carregado de modo a que o *MST Parser* possa analisar as frases de input.

Assume-se que essas frases são bem formadas de acordo com as regras de sintaxe da Língua Portuguesa.

Para cada frase do ficheiro de texto de input, o *Marcador Semântico* agrega informação sintáctica a cada palavra das frases através da ferramenta *LX-Suite*. De seguida o *MST Parser* calcula os arcos de dependências e respectivas etiquetas semânticas.

A ferramenta *Etiquetador* será responsável por obter os triplos de cada frase, e registar esses triplos (representações semânticas) de cada frase num documento RDF/XML.

Observemos com mais pormenor, cada módulo que constitui a ferramenta *Marcador Semântico*.

4.3.1 LX-Suite

O *LX-Suite* é uma ferramenta que foi descrita no Capítulo 3, Secção 3.2. Tem como objectivo agregar informação sintáctica a cada palavra da frase. Retomando o exemplo dado no capítulo que descreve esta ferramenta, considerando a seguinte frase:

A Maria tem razão.

A informação sintáctica agregada a esta frase, pela ferramenta *LX-Suite* seria:

```
<p> <s> A/DA#fs Maria/PNM tem/TER/V#pi-3s
    razão/RAZÃO/CN#gs ./PNT </s> </p>
```

As frases do texto de input são anotadas com informação sintáctica e passadas ao *MST Parser*.

4.3.2 Pipeline

Este *Pipeline* terá a mesma função que tem na ferramenta Web *LX Dep Parser*, com a excepção de não possuir a ferramenta *Desenha Árvores* que tem a função de produzir uma imagem do grafo de dependências de uma frase, pois para o *Marcador Semântico* não será necessário tal funcionalidade.

[2] modelo de treino obtido a partir do corpus de dependências com etiquetas semânticas

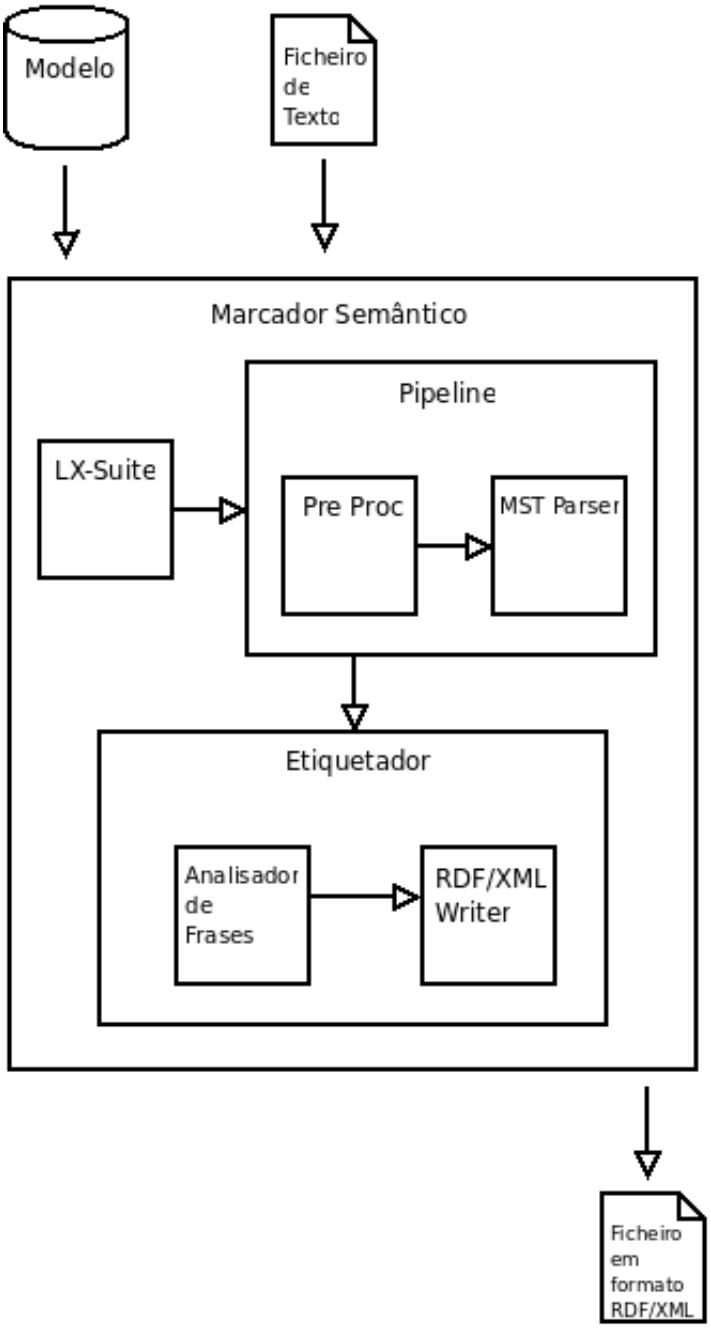


Figura 4.4: Diagrama da ferramenta *Marcador Semântico*

A ferramenta *MST Parser* recebe uma lista com as frases do texto de input anotadas. Através da ferramenta *Pre Proc*, estas frases são convertidas para o formato CoNLL 2006. Após este conversão, o *MST Parser* (*Parser*) analisa (parsing) as frases do texto, em formato CoNLL 2006 calculando os arcos de dependências e respectivas etiquetas gramaticais e semânticas. Deste modo, uma lista contendo as frases do texto de input convertidas para o formato CoNLL 2006 será passada à ferramenta *Etiquetador* que executa a ferramenta *Analisador de Frases* para que sejam extraídos os triplos resultantes da análise do *MST Parser* de cada frase do texto de input. Estes triplos serão escritos num ficheiro em formato RDF/XML pela ferramenta *RDF/XML Writer*.

4.3.3 Etiquetador

O *Etiquetador* recebe uma lista (ArrayList) com as frases do ficheiro de texto dado como input, frases essas que estão em formato CoNLL 2006, já com os arcos de dependências e respectivas etiquetas gramaticais e semânticas calculados.
A lista com as frases do texto de input já analisadas será passada ao *Analisador de Frases* para se obter os triplos correspondentes a cada frase do texto. Cada frase é processada obtendo os seus respectivos triplos, dando-se depois início à escrita destes triplos através da ferramenta *RDF/XML Writer*, para um documento RDF/XML.

Analisador de Frases

O processamento de cada frase inicia-se com a criação de um ArrayList "arrayConll" em que cada item da lista contém um objecto **ConllObj**. Este objecto **ConllObj** contém:

- **palavra** - A palavra.
- **posTag** - Etiqueta morfo-sintáctica da palavra.
- **id** - A posição da palavra na frase.
- **head** - A posição da palavra da qual esta palavra corrente depende.
- **depRel** - A etiqueta semântica da dependência com a palavra indicada em "head".
- **array** - Corresponde a um ArrayList do tipo Child chamado de **arrayFilhos**. Este array tem o objectivo de guardar os "filhos" de esta palavra. Ou seja, guarda todas as palavras que sejam "dependentes" desta palavra ao qual corresponde o objecto **ConllObj**. De referir que no parsing de dependências/formato CoNLL 2006, uma palavra pode ter várias palavra dependentes associadas. No entanto apenas pode ser dependente de uma e uma só palavra.

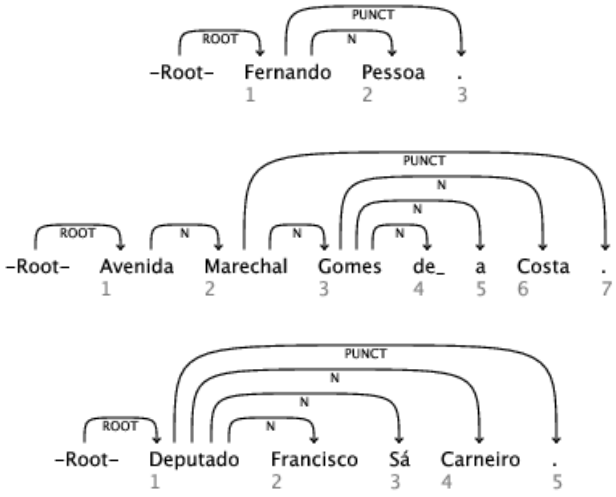
  Torna-se necessária esta estrutura de dados para guardar a informação sobre os "filhos" de uma palavra uma vez que no formato CoNLL 2006 não temos acesso a

tal informação. Como já foi escrito anteriormente, este array contém objectos do tipo "Child". Um Objecto do tipo "Child" contém:

- **filho** - A posição da palavra na frase.
- **relFilho** - A etiqueta semântica que define o tipo de dependência com a palavra-pai.

Na construção deste arrayConll, é ainda executado um passo de pré-processamento que dará uma ajuda ao algoritmo encarregue de capturar os triplos evitando passos na pesquisa, no arrayConll.

Assim sendo, é feito um pré-processamento sobre um conjunto de palavras que constituam "Entidades Nomeadas". As entidade nomeadas são expressões que designam uma entidade que em princípio terá uma designação única. Esta designação é constituída por um ou mais nomes comuns ou próprios, como por exemplo "Fernando Pessoa", "Avenida Marechal Gomes da Costa", ou "Deputado Francisco Sá Carneiro". As estruturas de dependências de estas entidades nomeadas são as seguintes, respectivamente:



Nestas entidades nomeadas é possível verificar que os nomes que as constituem estão relacionados entre si com a etiqueta gramatical "N" (Nome). O objectivo deste pré-processamento é juntar todas as palavras que constituem a entidade nomeada na posição da primeira palavra da mesma entidade nomeada:

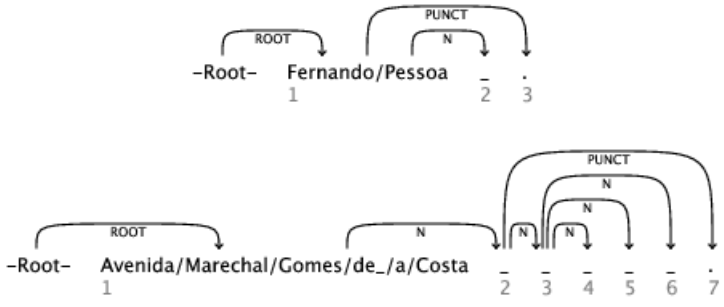


Figura 4.5: Exemplo de relação semântica de modificador de Modo.

Uma vez que, a não ser a primeira palavra da entidade nomeada, as restantes palavras da entidade nomeada não se envolverão em nenhuma outra relação de dependência com outras palavras a não ser com palavras dentro da entidade nomeada em questão, este pré-processamento poupa ao algoritmo de captura do triplos, presente na ferramenta *Analisador de Frases*, alguns passos de execução.

Tendo o **arrayConll** devidamente preenchido, começaremos a percorrer o array a partir da posição da palavra que será a raiz da frase. Chamemos de palavra-pai à palavra que está a ser avaliada (Núcleo), e chamemos de palavra-filho à palavra que possui uma relação de dependência com a palavra-pai que está a ser avaliada (Dependente).

Dada uma palavra-pai, o que o *Analisador de Frases* executa é uma série de verificações à etiqueta do arco de dependências em relação à palavra-filho.
No geral, a etiqueta do arco de dependências entre a palavra-pai e a palavra-filho é verificada assim como, em alguns casos, as etiquetas morfo-sintácticas da palavra-pai e/ou palavra-filho de modo a se obter o triplo que consiste: na palavra-pai, etiqueta do arco de dependência e palavra-filho. Caso o triplo seja obtido, repete-se o processo recursivamente, para a palavra-filho. Quando não existir mais palavras que sejam "filhos" da palavra que é a raiz da frase, ou por outras palavras, quando o **arrayFilhos** da palavra raiz, for percorrido, a execução desta pesquisa por triplos termina.

Como já referido, de modo a obter os triplos são efectuadas algumas verificações às etiquetas dos arcos de dependências, e em alguns casos, à etiqueta morfo-sintáctica da palavra filho. Passemos então a descrever essas verificações.

Registo dos Triplos

As verificações das etiquetas dos arcos de dependências que são executadas de modo a fazer triplos para serem registados no formato RDF/XML, são as seguintes:

- Verificação se a etiqueta do arco de dependência entre a palavra-pai e a palavra-

Figura 4.6: Exemplo de uma relação de dependência gramatical de predicado.

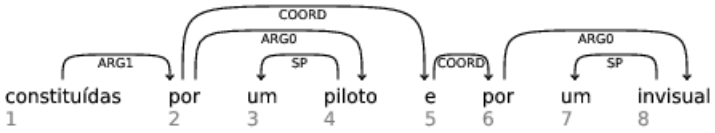


Figura 4.7: Exemplo de uma conjunção coordenativa.

Figura 4.8: Exemplo da relação semântica “ARG1”.

Figura 4.9: Exemplo da relação semântica “ARG0”.

filho, é um modificador. Caso a palavra-filho tenha uma relação de dependência do tipo "modificador" com a palavra-pai, é desde logo registado o triplo com a palavra-pai, arco de dependência, palavra-filho.
Observando a Figura 4.5, é possível constatar que o triplo obtido seria:

(maçã, M-MNR, sem)

- Verificação se a etiqueta do arco de dependência entre a palavra-pai e a palavra-filho, é um predicado. Caso a palavra-filho seja um verbo, particípio passado em tempos compostos, ou um particípio passado que não forme tempos compostos, e tenha uma relação de dependência do tipo "predicado", com a palavra-pai, é desde logo registado o triplo.
  Observando a Figura 4.6 é possível constatar que o triplo obtido seria:

  (Foi, PRD, examinado)

- Verificação se a etiqueta do arco de dependência entre a palavra-pai e a palavra-filho, é um Sintagma preposicional ou nominal, ou uma coordenação. Caso a palavra-filho seja uma conjunção, é registado o triplo que envolva a palavra-pai da palavra-pai cujo palavra-filho esta a ser verifica, a etiqueta do arco de dependência, e a palavra-filho (Conjunção coordenativa).
  Interessa referir que as conjunções coordenativas são as visadas por esta verificação.
  Observando a Figura 4.7 é possível constatar que verificando a etiqueta do arco de dependência entre as palavras "por" e "e", o triplo obtido seria:

  (constituídas, COORD, e)

  e posteriormente seriam obtidos os triplos:

  (e, COORD, piloto)
  (e, COORD, invisual)

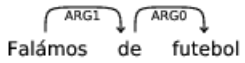
- Verificação se a etiqueta do arco de dependência entre a palavra-pai e a palavra-filho, corresponde a Argumento (que não seja igual a "ARG0"). Caso a palavra filho seja um nome comum, nome próprio, título pessoal ou pronome pessoal, é desde logo registado o triplo contendo a palavra-pai, o arco de dependência, e a palavra-filho.
  Observando a Figura 4.8 é possível constatar que o triplo obtido seria:

  (bebeu, ARG1, água)

- Verificação se a etiqueta do arco de dependência corresponde a "ARG0", entre a palavra-pai e a palavra-filho. Se a palavra-filho for um nome comum, nome próprio, título pessoal (por exemplo: Sr. Engº), pronome pessoal (eu, tu ele, nós, vós, eles) ou verbo, e a palavra-pai for por exemplo uma preposição modificadora, é desde logo obtido o triplo entre a palavra-pai e palavra-filho, com o arco de Dependencia "ARG0". casos pontuais.
  Observando a Figura 4.9 é possível constatar que os triplos obtidos seriam:

  (assuntos, M-ADV, de)
  (de, ARG0, homem)

- De resto, são executadas algumas verificações às etiquetas morfo-sintácticas das palavras-filho de modo a despistar a captura de determinados triplos. Por exemplo, poderemo-nos deparar com um caso em que exista um preposição não modificadora, entre a raiz da palavra (uma palavra-pai que seja um verbo) e um nome comum (que seja a palavra-filho):

  Considerando o excerto de uma frase acima, este tipo de verificação tem o objectivo de passarmos pela preposição observando a sua palavra-filho ("futebol"), ou seja, uma vez que a preposição "de" não é modificadora do verbo falar, iremos querer verificar quais as palavras-filhos da preposição. Sendo que o nome comum "futebol" é a palavra-filho da preposição, e uma vez que a preposição que não faz qualquer tipo de modificação ao verbo falar, proceder-se-á ao registo do triplo

  (Falámos, ARG1, futebol)

  em que futebol será "o assunto de que se fala". Por outras palavras, "futebol" será o primeiro argumento do verbo "Falámos".

De referir que os triplos obtidos são guardados numa "hashMap" em que cada chave da "hashMap" irá corresponder a um Sujeito do conjuntos de triplos de cada frase. A cada Sujeito está associada uma lista do tipo "PredObj" (Predicado & Objecto) em que cada posição da lista contém o Predicado e Objecto, do triplo que corresponde ao Sujeito. É necessário referir um Sujeito pode ser o Sujeito de vários triplos da mesma frase. Os triplos são assim guardados nesta estrutura de informação, pois facilitará a escrita destes triplos no documento RDF/XML.

Após a obtenção dos triplos das frases do texto, inicia-se a escrita desses mesmos triplos para um ficheiro RDF/XML.

4.3.4 RDF/XML Writer

Após a recolha dos triplos de cada frase do ficheiro de texto dado como input, o hashMap contendo os triplos de cada frase é analisado de modo a que se codifique a representação semântica de cada frase (triplos) na linguagem de marcação RDF/XML. Segue-se uma explicação de como se constrói um grafo RDF/XML, após a obtenção dos triplos da frase.

Grafo RDF/XML

Um grafo RDF/XML consiste num conjunto de triplos definidos com base na sintaxe da linguagem RDF, encapsulado nos termos da linguagem XML. Triplos esse que como já vimos, são constituídos por:

- **Nó Sujeito** - que pode ou não ser identificado por uma referência RDF URI. Em caso de não ser identificado por uma referência RDF URI, o nó será um "blank node".
- **Predicado** - que apenas pode ser uma referência RDF URI.
- **Objecto** - que poderá um nó "blank node", um nó identificado por uma referência RDF URI ou um "literal".

Observando a Figura 4.10, é possível ver um grafo RDF/XML que de uma maneira geral representa uma `camisa` com o `tamanho 38`, de `cor preta`.
Este grafo é constituído por dois triplos:

- **(camisa, cor, preto)** - Constituído por um nó Sujeito identificado por um referência RDF URI (a string `camisa`), por um nó Objecto também identificado por uma referência RDF URI (a string `preto`) e por um Predicado cuja o nome (referência RDF URI) é `cor`.
- **(camisa, tamanho, 38)** - Constituído por o nó Sujeito `camisa`, por um Objecto "literal" (mais concretamente um "plain literal" com a string `38`), e por um Predicado com o nome de `tamanho`.

De modo a definir o grafo da Figura 4.10 num documento de linguagem de marcação RDF/XML, o seguinte código é criado:

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns/"
    xmlns:caracteristica="http://www.camisolasnanet.com/
        features-roupa#">

    <rdf:Description rdf:about="camisa">
```

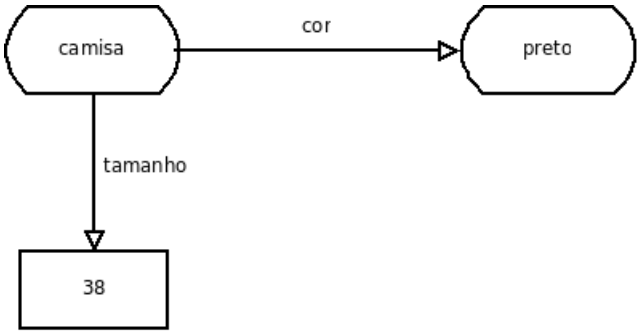


Figura 4.10: Grafo RDF/XML que define a camisola azul de tamanho 38.

```
        <caracteristica:tamanho>38</caracteristica:tamanho>
        <caracteristica:cor rdf:resource="preto"/>

    </rdf:Description>

</rdf:RDF>
```

Explicando passo a passo o código acima, de modo a se construir um documento RDF/XML deve-se definir o nó raiz:

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

</rdf:RDF>
```

O nó raiz <rdf:RDF> </rdf:RDF> do documento RDF/XML, engloba todo(s) o(s) grafo(s) de um documentos RDF/XML. Por omissão, este elemento nunca é representado num grafo RDF/XML.
Também podemos observar o "Qualified Name" "rdf" que está declarado no espaço de nomes XML com a descrição "http://www.w3.org/1999/02/22-rdf-syntax-ns#" que é atribuída por omissão, a este elemento em todos os documentos RDF/XML. O nó raiz do documento (<rdf:RDF> </rdf:RDF>) e o espaço de nomes XML do código acima (xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#") formam a "raiz" de qualquer documento RDF/XML.

O espaço de nomes XML é uma colecção de elementos identificados por referências URI (Uniform Resource Identifier), que é usado para definir univocamente "Qualified Names" (de referir que uma referência URI é semelhante a uma referência RDF URI, na medida em que duas URI são comparadas caracter a caracter. Enquanto as referências RDF URI

definem os nomes dos nós, as referências URI definem os nomes dos "Qualified Names" no espaço de nomes XML).
Num documento RDF/XML, os "Qualified Names", servem para definir relações entre elementos RDF (referências RDF URI, "Blank nodes" e "Literals") constituindo os triplos RDF/XML.
Os "Qualified Names" são declarados tendo como prefixo a etiqueta "xmlns:", um nome local (por exemplo "caracteristica") e um nome dentro do espaço de nomes que será uma referência URI (por exemplo http://www.camisolasnanet.com/features-roupa#).

De modo a definir Predicados únicos, uma maneira de efectuar tal procedimento é permitir (por omissão) que a máquina ou o interpretador de RDF/XML, ao analisar o documento RDF/XML, concatene o nome do espaço de nomes do "Qualified Name" com o atributo definido pelo "Qualified Name" (por exemplo, se o nome do "Qualified Name" for "www.exemplo.com" e se o nome do "Qualified Name" for "coisa" - `xmlns:coisa="www.exemplo.com"` - e definirmos um Predicado através do atributo "coisa:tem", então o Predicado ficará com o nome de 'www.exemplo.comtem".
Como não será necessário tal procedimento para diferenciar Predicados, nem neste exemplo nem para o documento RDF/XML que o *Marcador Semântico* produz, todos os nomes do espaço de nomes dos "Qualified Names" terão como último caracter "#", para evitar a concatenação do nome do Predicado com o nome do "Qualified Name" que define esse mesmo Predicado.

Para criarmos o triplo **(camisa, tamanho, 38)** é necessário criarmos o nó `camisa`. Para declararmos um nó em RDF/XML, utilizamos o elemento definido por omissão <rdf:Description>. De modo a que o nó não seja um "blank node", é necessário atribuir uma referência RDF URI com o atributo "rdf:about", que de um modo geral será o nome do nó. Como tal, o seguinte código mostra a criação do nó `camisa`:

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

    <rdf:Description rdf:about="camisa">

    </rdf:Description>

</rdf:RDF>
```

Para concretizar o triplo **(camisa, tamanho, 38)** é necessário criar um Predicado. Para tal, declara-se o "Qualified Name" `caracteristica` no espaço de nomes:

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
```

```
    xmlns:caracteristica="http://www.camisolasnanet.com/
        features-roupa#">

    <rdf:Description rdf:about="camisa">

    </rdf:Description>

</rdf:RDF>
```

De referir que não será de estranhar que o nome do "Qualified Name" `caracteristica` seja um URL pois a linguagem de marcação RDF/XML foi concebida para atribuir significado a recursos da Web e relacionar esses mesmos recursos. O que não significa que não se possa criar elementos num documento RDF/XML e atribuir relações entre os mesmos. Deste modo, a string (referência URI) que é o nome do "Qualified Name" `caracteristica` não tem de ter o formato obrigatório de um URL.

Com o "Qualified Name" `caracteristica` definido, podemos definir o Predicado `tamanho` para associar o "literal" `38`, construindo o triplo **(camisa, tamanho, 38)**:

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:caracteristica="http://www.camisolasnanet.com/
        features-roupa#">

    <rdf:Description rdf:about="camisa">

        <caracteristica:tamanho>38</caracteristica:tamanho>

    </rdf:Description>

</rdf:RDF>
```

De referir que, dentro da declaração do nó `camisa`, podemos escrever <caracteristica:tamanho>38</feature:caracteristica> ou <caracteristica="38"/> para associarmos ao nó `camisa`, o "literal" `38` com o Predicado `tamanho`.

Para que o triplo **(camisa, cor, preto)** seja criado, podemos utilizar novamente o "Qualified Name" `caracteristica` para definirmos o Predicado `cor` (caracteristica:cor). Como queremos associar o nó `preto` ao nó `camisa`, podemos utilizar o atributo "rdf:resource", dentro da declaração do nó `camisa`, que permite associar um nó já constituído por uma referência RDF URI sem termos de criar o nó através do elemento "<rdf:Description>" e do atributo "rdf:about". Como tal, passamos a ter o seguinte código:

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:caracteristica="http://www.camisolasnanet.com/
        features-roupa#">

    <rdf:Description rdf:about="camisa">

        <caracteristica:tamanho>38</caracteristica:tamanho>
        <caracteristica:cor rdf:resource="preto"/>

    </rdf:Description>

</rdf:RDF>
```

em que os triplos **(camisa, tamanho, 38)** e **(camisa, cor, preto)** estão definidos.

Representação Semântica em RDF/XML

De modo a construirmos um documento RDF/XML com a representação semântica de frases de um dado texto, o mecanismo do exemplo anterior será usado. Ou seja, serão marcados/codificados os triplos que fazem parte da representação semântica de uma frase, em RDF/XML.
Considerando novamente a frase: "A Maria tem razão." e os triplos que representam a semântica da mesma:

(Root, ROOT, tem)
(tem, ARG1, Maria)
(tem, ARG2, razão)

iremos construir um documento RDF/XML para registar a representação semântica de um frase.

Para a construção de um documento RDF/XML, para além do elemento raiz do ficheiro RDF/XML ter de ser declarado, como já foi visto, vamos também declarar os "Qualified Names" (as etiquetas semânticas `ROOT`, `ARG1` e `ARG2`) no espaço de nomes:

```
<rdf:RDF
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:ARG1="Argument_1#"
     xmlns:ARG2="Argument_2#"
     xmlns:ROOT="Root_Sentence#"
>
```

De referir na fase de construção do grafo RDF/XML, o *Marcador Semântico* percorre-se a hashMap com os triplos de uma dada frase do texto de input, obtendo cada chave de

modo a declarar o nó Sujeito. De seguida associa-se ao nó Sujeito os nós Objecto (definindo um Predicado com um dado "Qualified Name" definido no espaço de nomes e o atributo reservado "rdf:resource" para associarmos o nó Objecto) com os quais o nó Sujeito constitui o(s) triplo(s).
Definindo os Predicados de cada triplo (`Root_Sentence`, `Argument_1` e `Argument_2`) triplos da frase e associando as palavras da frase com os Predicados:

- **(Root, ROOT, tem)** - nó `Root` associado ao nó `tem` através do Predicado `Root`.
- **(tem, ARG1, Maria)** - nó `tem` associado ao nó `Maria` através do Predicado `ARG1`.
- **(tem, ARG2, razão)** - nó `tem` associado ao nó `razão` através do Predicado `ARG2`.

teríamos o seguinte código:

```
<?xml version="1.0"?>
<rdf:RDF
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:ARG1="Argument_1#"
     xmlns:ARG2="Argument_2#"
     xmlns:ROOT="Root_Sentence#"
>

<rdf:Description rdf:about="Root">
     ROOT:Root rdf:resource="tem"
</rdf:Description>

<rdf:Description rdf:about="tem">
     ARG1:Argument_1 rdf:resource="Maria/>"
     ARG2:Argument_2 rdf:resource="razão"/>
</rdf:Description>

</rdf:RDF>
```

Uma vez que iremos registar triplos de várias frases, e que uma mesma palavra poderá estar presente em duas frases ou que duas palavras iguais na mesma frase poderão ser pertencer a triplos distintos, teremos de reforçar a estrutura do código do documento RDF/XML por forma a desambiguar palavra iguais em triplos distintos, pois dois ou mais nós com o mesmo RDF URI Reference (rdf:about="palavra"), serão interpretados como sendo o mesmo nó.

Como tal foram definidas as seguintes regras para construir uma estrutura coerente e precisa para o documento RDF/XML:

- Referenciar os nós das palavras, não pela palavra em si mas pela posição (prefixada com a frase em questão, no documento RDF/XML) da palavra na frase:

```
<rdf:Description about="Sentence_N-Word_3">
     <ARG1:Argument_1 rdf:resource="Sentence_N-Word_2\>"
</rdf:Description>
```

- Criar um "Qualified Name" `Value` de modo a definir o Predicado `Word`. Predicado esse que estabelece uma relação entre o nó com a posição da palavra ("Sentence_X-Word_Y") e o valor da palavra em si (palavra essa que no código do documento RDF/XML será um "plain literal"):

```
<rdf:Description about="Sentence_N-Word_3"
     Value:Word="tem">

     <ARG1:Argument_1 rdf:resource="Sentence_N-Word_2"
          Value:Word="Maria"/>
</rdf:Description>
```

  De referir que `<Value:Word="Maria"/>` é equivalente a `<Value:Word>Maria</Value:Word>`.

- Definir um nó para uma dada frase N que forme um triplo com o nó que corresponde à palavra "root" da frase N. Triplo esse que será formado através do Predicado `Root`:

```
<rdf:Description about="Sentence_N">
     <ROOT:Root rdf:resource="tem"/>
</rdf:Description>
```

Estas regras garantem que cada palavra tem um identificador diferente. Contudo, não é realizada qualquer desambiguação de referencias. Este tema será discutido mais à frente.

Tendo em conta que seja quais forem as frases, os "Qualified Names" `ROOT` e `Value` terão de ser declarados no espaço de nomes para definirmos o predicados `Root` e `Word`. Assim sendo, para a frase:

será construído o grafo RDF/XML (Figura 4.11) cujo código é:

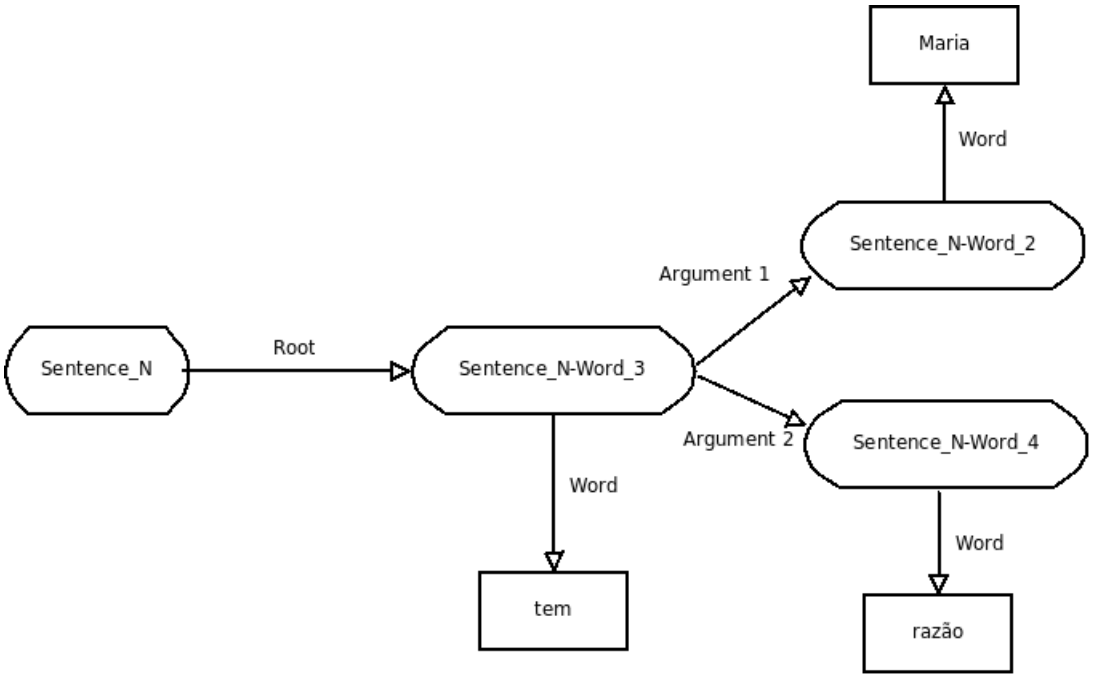


Figura 4.11: Grafo RDF/XML da frase: “A Maria tem razão.”

```
<?xml version="1.0"?>

<rdf:RDF
<rdf:RDF
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:ARG1="Argument_1#"
     xmlns:ARG2="Argument_2#"
     xmlns:ROOT="Root_Sentence#"
     xmlns:Value="Word_Value#"
>

    <rdf:Description rdf:about="Sentence_N-Word_3"
        Value:Word="tem">

        <ARG1:Argument_1 rdf:resource="Sentence_N-Word_2"
            Value:Word="Maria"/>

        <ARG2:Argument_2 rdf:resource="Sentence_N-Word_4"
            Value:Word="razão"/>

    </rdf:Description>
```

```
    <rdf:Description rdf:about="Sentence_N">

        <ROOT:Root rdf:resource="Sentence_N-Word_3"
            Value:Word="tem"/>

    </rdf:Description>

</rdf:RDF>
```

No entanto, existem outras possibilidades de estruturar a representação semântica de uma frase. Por exemplo, olhando pro grafo da Figura 4.11, podemos ver que os "literals" que representam o valor de cada palavra de uma frase poderiam ser um nó, o que não faria sentido pois voltaríamos à questão de termos nós com o mesmo nome, representando palavras de diferentes fases (assumindo que o valor da palavra seria a referência RDF URI do nó) pois como já foi referido, duas referências RDF URI são iguais quando comparadas caracter a caracter).
Poderíamos estruturar o grafo RDF/XML da Figura 4.11 de modo a que o nó da frase `Sentence_N` se ligasse a todos os outros nós pertencentes ao grafo (`Sentence_N-Word_2`, `Sentence_N-Word_3` e `Sentence_N-Word_4`), o que seria redundante, pois a partir do nó `Sentence_N` (uma vez que está ligado ao nó que corresponde à "root" da frase "A Maria tem razão."), é possível chegar aos restantes nós que constituem a representação semântica da frase em questão.

Portanto, a estrutura do grafo RDF/XML da Figura 4.11, representa uma estrutura simples e adequada para codificar a representação semântica de cada frase de um texto processado pela ferramenta *Marcador Semântico*. Em que, para além da estrutura do grafo RDF/XML de cada frase (Figura 4.11) permitir a desambiguação de palavras iguais em diferentes frases no documento RDF/XML, cada nó `Sentence_N` representa uma frase no documento RDF/XML, estabelece uma ligação com o nó da palavra, que é a "root" da representação semântica da frase em questão. O que torna possível no grafo RDF/XML, chegarmos a todas as outras palavras (nós da palavras e consequentemente os "literals" que representam o valor da palavra em questão) da representação semântica.

Desambiguação de Referências

Consideremos, por exemplo, a frase "Falei com a Maria e a Maria disse:". Podemos ver que ambas as palavras "Maria" têm o mesmo referente.
Outro exemplo de uma frase que apresenta este tipo de problema é a frase "Conversei com o Francisco e o Rui, e ele disse:". Nesta frase é possível observar que a palavra "ele" refere-se ao "Francisco" ou ao "Rui".

De acordo com as regras descritas anteriormente, todas as palavras terão um identificador diferente ainda que correspondam ao mesmo referente. A tarefa de desambiguação de referências—que constitui um vasto problema dentro da área do Processamento de Linguagem Natural, para o qual ainda não existe uma solução concreta—seria da responsabilidade de uma ferramenta subsequente, estando portanto fora do âmbito desta dissertação.

4.4 Considerações Finais

Neste capítulo foi explicada a ferramenta *Marcador Semântico* assim como os módulos que integram a ferramenta. Com esta ferramenta torna-se possível o registo da representação semântica da informação textual (escrita na Língua Portuguesa) em Linguagem de marcação RDF/XML.

No próximo Capítulo será apresentada a conclusão desta dissertação.

Capítulo 5

Conclusão

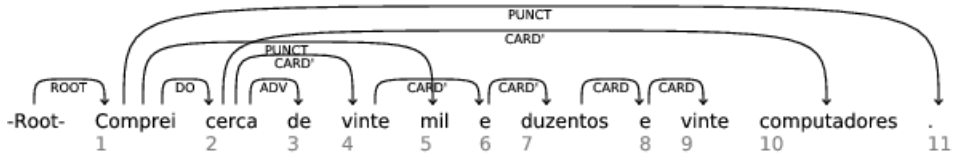
5.1 Comentários Finais

Nesta dissertação foi apresentada a ferramenta *Marcador Semântico* capaz de obter uma representação semântica de um texto.
Também foi apresentada a ferramenta Web *LX Dep Parser* que devolve uma representação das relações gramaticais, entre as palavras de uma frase.
Estas ferramentas são definidas à custa de um parser de dependências gramatical chamado de MST Parser, calcula os arcos de dependências e respectivas etiquetas.
Com corpus de dependências do Grupo NLX, com etiquetas semânticas atinge o resultado de 0,8533 para a métrica LAS, o que significa que para um corpus com etiquetas semânticas em Português, é o melhor resultado publicado para o Português. Já com o mesmo corpus mas apenas anotado com etiquetas gramaticais, o resultado obtido para a métrica LAS é de 0,8771, o que é o melhor resultado observado para corpus de dependências com apenas etiquetas gramáticas, em Língua Portuguesa.

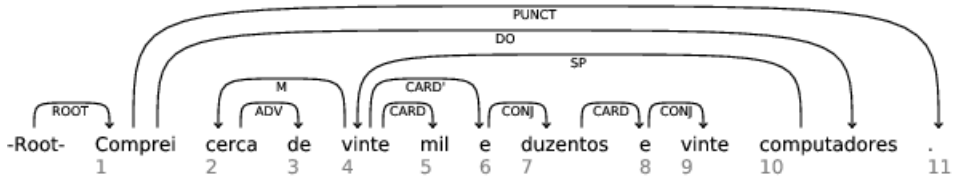
Como já referido, a ferramenta *Marcador Semântico* fornece uma representação semântica de um texto de input. Representação essa que é codificada em linguagem de marcação RDF/XML. Uma das aplicações possíveis para este tipo de representações semânticas de uma dada frase (e consequentemente de um texto), seria um motor de busca, que em vez de procurar uma frase por ocorrência num qualquer texto, analisaria essa mesma frase para obter os seus triplos, pesquisando esses mesmo os triplos em repositórios que contivessem documentos RDF/XML, com a representação semântica de textos pertencentes a variadíssimas páginas web, escritas em Português.
Deste modo seria possível encontrar uma ocorrência da frase pela qual se efectuava a busca, através da sua semântica, o que iria de encontro ao significado da informação que o utilizador estivesse à procura.

Apesar da efectividade de esta ferramenta em extrair os triplos de uma frase, é necessário referir que o parser de dependências MST, possui limitações. Os resultados obtidos para as métricas LAS: 0,8533 e UAS: 0,9309, mostram que este parser irá cometer

erros na atribuição de arcos de dependências e suas respectivas etiquetas gramaticais e semânticas, pois obter 100% na métrica LAS é algo que ainda está distante de acontecer, seja qual for o parser de dependências utilizado ou o corpus de dependências com que se treina o parser de dependências. Contudo, é necessário referir que o corpus de dependências com etiquetas semânticas do Grupo NLX, é um corpus relativamente pequeno com mil duzentos e quatro frases (com trinta e um tipos de dependências), capaz de atingir excelentes resultados. Corpus esse que desde o dia um da elaboração desta dissertação, foi alvo de várias actualizações em relação a anotações dos arcos de dependências de algumas frases:



onde da actualização da anotação desta frase resultou uma frase não projectiva:



Outra situação a referir é o facto de não se ter efectuado a verificação da extracção do triplos para todas as frases do corpus, ou seja, não se verificou se esta ferramenta retira correctamente os triplos de todas as frases anotadas, do corpus de dependências do NLX. Tal não foi efectuado porque era necessária a verificação manual dos triplos, com cada uma frase do corpus, pois não existe nenhum corpus dourado de triplos, das frases presentes no corpus, para fazer testes de validação. O que foi feito foi a escolha de alguma frases que contivessem, não só um bom número de arcos de dependências, mas também uma boa selecção de tipos de dependências como por exemplo, a conjunção coordenativa, de modo a conseguir generalizar o funcionamento da ferramenta *Marcador Semântico* para quaisquer tipos de relações de dependência.

Apesar de algumas limitações devido à quantidade de informação disponível para treinar o parser de dependências, esta ferramenta apresenta-se como uma inovação na área do processamento da linguagem natural na língua Portuguesa, e como um ponto de partida para o armazenamento de representações semânticas da informação presente na web. Representações semânticas essas, que poderão ser alvo de agentes autómatos ou por motores de busca semânticos.

5.2 Trabalho Futuro

São algumas as melhorias a considerar para melhorar o desempenho do *Marcador Semântico*. Em primeiro lugar, uma actualização ao corpus de dependências do Grupo NLX seria desejável, pois o aumento de quantidade de informação com que se treina o parser de dependências (neste caso o MST Parser) poderá trazer melhorias quanto aos resultados de parsing. Em segundo lugar, correr novamente um teste entre os parsers de dependências com os melhores resultados (Malt Parser e MST Parser) e verificar qual o parser de dependências que produz os melhores resultados (efectuando novamente um teste "Ten Fold Cross Validation"). Caso seja o Malt Parser que produza os melhores resultados, será necessário integrar o mesmo parser na ferramenta *Marcador Semântico*.

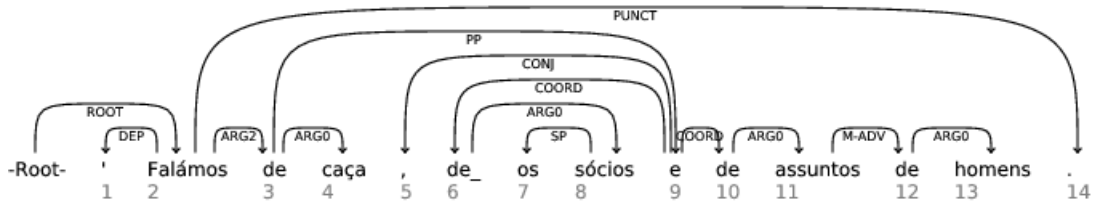
A longo prazo, e certamente, a actualização a efectuar ao *Marcador Semântico*, seria a construção de uma ferramenta que corresse sobre o próprio *Marcador Semântico*, capaz de identificar referências iguais.
Retomando o exemplo "Falei com a Maria e a Maria disse:", seria de enorme interesse conceber uma ferramenta que fosse capaz de identificar que ambas palavras "Maria" dizem respeito à mesma entidade, melhorando assim a estrutura do documento RDF/XML, produzido pelo *Marcador Semântico*.

Uma ideia a ser concretizada, também a longo prazo, seria a integração da ferramenta *Marcador Semântico* num motor de busca. Este motor de busca "semântico" receberia uma frase com a qual extrairia uma representação semântica (utilizando para isso o *Marcador Semântico*). Com a representação semântica extraída do input, seria efectuada uma busca em repositórios de representações semânticas com a informação da Web registada (sob a forma de documentos RDF/XML), com o objectivo de devolver os sítios da Web onde se encontraria a informação procurada.

Apêndice A

Triplos de algumas Frases do Corpus de Dependências

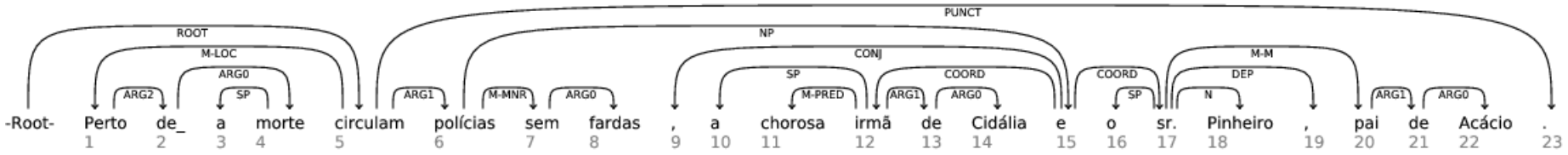
Para a Frase (0):



os triplos são:

(Falámos,	ARG2,	e)
(assuntos,	M-ADV,	de)
(de,	ARG0,	homens)
(e,	COORD,	caça)
(e,	COORD,	sócios)
(e,	COORD,	assuntos)
(Root,	ROOT,	Falámos)

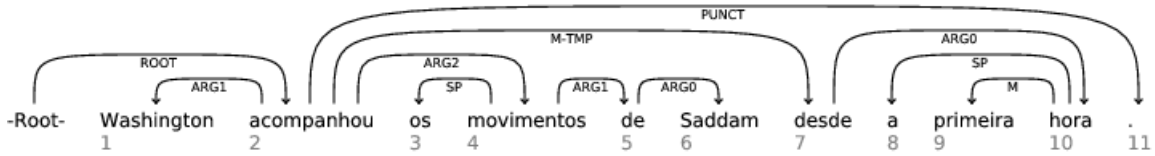
Para a Frase (1):



os triplos são:

(pai,	ARG1,	Acácio)
(sr.Pinheiro,	M-M,	pai)
(circulam,	M-LOC,	Perto)
(circulam,	ARG1,	e)
(Root,	ROOT,	circulam)
(e,	COORD,	polícias)
(e,	COORD,	irmã)
(e,	COORD,	sr. Pinheiro)
(Perto,	ARG2,	morte)
(irmã,	M-PRED,	chorosa)
(irmã,	ARG1, Cidália)	
(sem,	ARG0,	fardas)
(polícias,	M-MNR,	sem)

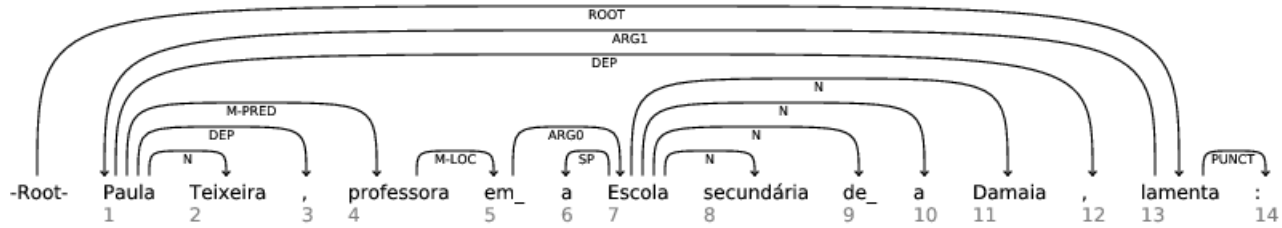
Para a Frase (2):



os triplos são:

(desde,	ARG0,	hora)
(hora,	M,	primeira)
(movimentos,	ARG1,	Saddam)
(acompanhou,	ARG1,	Washington)
(acompanhou,	ARG2,	movimentos)
(acompanhou,	M-TMP,	desde)
(Root,	ROOT,	acompanhou)

Para a Frase (3):



os triplos são:

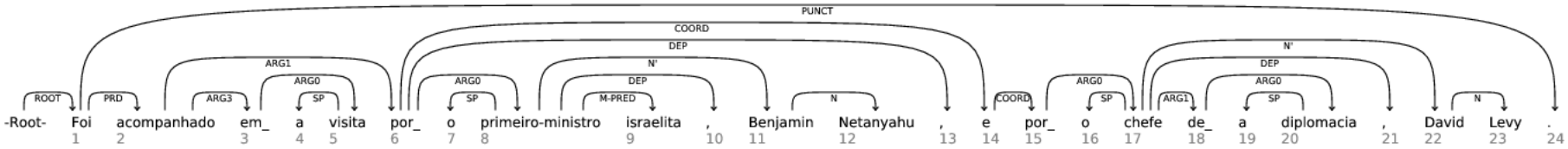
(professora,	M-LOC,	em_)
(em_,	ARG0,	Escola_secundária_de__a_Damaia)
(lamenta,	ARG1,	Paula_Teixeira)
(Paula_Teixeira,	M-PRED,	professora)
(Root,	ROOT,	lamenta)

Para a Frase (4):

os triplos são:

(e,	COORD,	PSD)
(e,	COORD,	PS)
(trabalhos,	ARG1,	Constituição)
(Root,	ROOT,	desaceleram)
(desaceleram,	ARGA,	e)
(desaceleram,	ARG1,	trabalhos)

Para a Frase (5):



os triplos são:

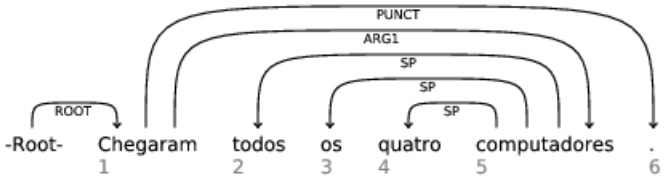
(e,	COORD,	primeiro-ministro)
(e,	COORD,	chefe)
(Root,	ROOT,	Foi)
(chefe,	ARG1,	diplomacia)
(chefe,	N',	David_Levy)
(acompanhado,	ARG3,	visita)
(acompanhado,	ARG1,	e)
(Foi,	PRD,	acompanhado)
(primeiro-ministro,	M-PRED,	israelita)
(primeiro-ministro,	N',	Benjamin_Netanyahu)

Para a Frase (6):

os triplos são:

(fugiam,	ARG1,	vítimas)
(fugiam,	M-ADV,	não)
(fugiam,	ARG2,	normalidade)
(Root,	ROOT,	fugiam)

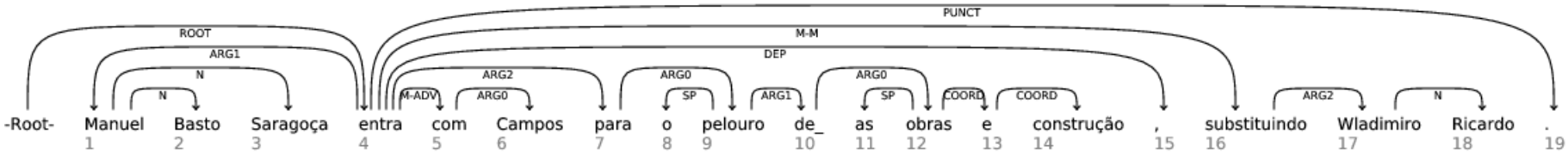
Para a Frase (7):



os triplos são:

(computadores,	SP,	todos)
(computadores,	SP,	quatro)
(Chegaram,	ARG1,	computadores)
(Root,	ROOT,	Chegaram)

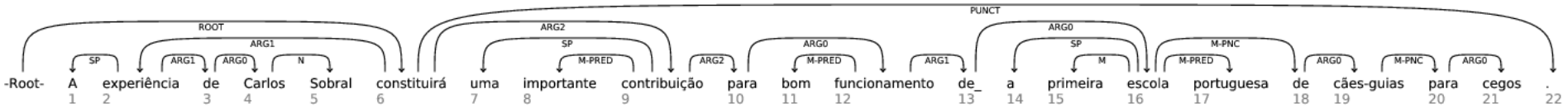
Para a Frase (8):



os triplos são:

(pelouro,	ARG1,	e)
(e,	COORD,	obras)
(e,	COORD,	construção)
(entra,	ARG1,	Manuel_Basto_Saragoça)
(entra,	M-ADV,	com)
(entra,	ARG2,	pelouro)
(entra,	M-M,	substituindo)
(Root,	ROOT,	entra)
(substituindo,	ARG2,	Wladimiro_Ricardo)
(com,	ARG0,	Campos)

Para a Frase (9):



os triplos são:

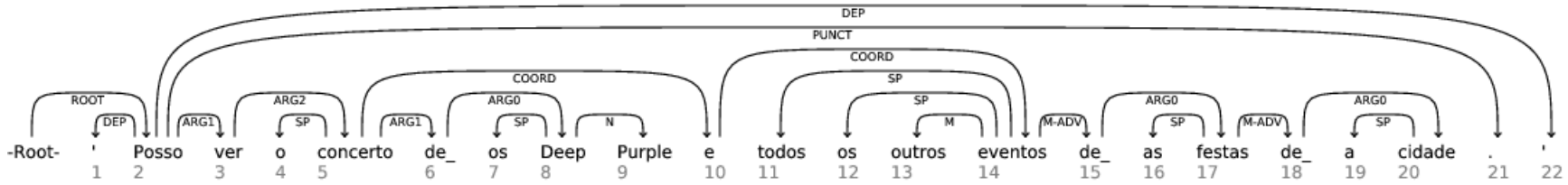
(constituirá,	ARG1,	experiência)
(constituirá,	ARG2,	contribuição)
(para,	ARG0,	cegos)
(de,	ARG0,	cães-guias)
(Root,	ROOT,	constituirá)
(escola,	M,	primeira)
(escola,	M-PRED,	portuguesa)
(escola,	M-PNC,	de)
(cães-guias,	M-PNC,	para)
(experiência,	ARG1,	Carlos_Sobral)
(contribuição,	M-PRED,	importante)
(contribuição,	ARG2,	funcionamento)
(funcionamento,	M-PRED,	bom)
(funcionamento,	ARG1,	escola)

Para a Frase (10):

os triplos são:

(manutenção,	ARG2,	escola)
(manutenção,	M-TMP,	até)
(até,	ARG0,	1999)
(fundos,	M-PRED,	comunitários)
(asseguram,	ARG1,	fundos)
(asseguram,	ARG2,	manutenção)
(Root,	ROOT,	asseguram)

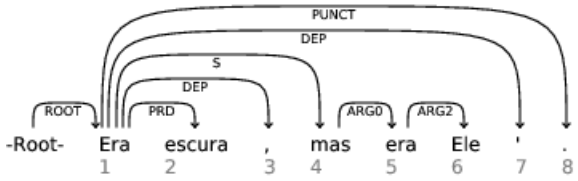
Para a Frase (11):



os triplos são:

(de_,	ARG0,	cidade)
(de_,	ARG0,	festas)
(Posso,	ARG1,	ver)
(eventos,	SP,	todos)
(eventos,	M,	outros)
(eventos,	M-ADV,	de_)
(Root,	ROOT,	Posso)
(concerto,	ARG1,	Deep_Purple)
(festas,	M-ADV,	de_)
(e,	COORD,	concerto)
(e,	COORD,	eventos)
(ver,	ARG2,	e)

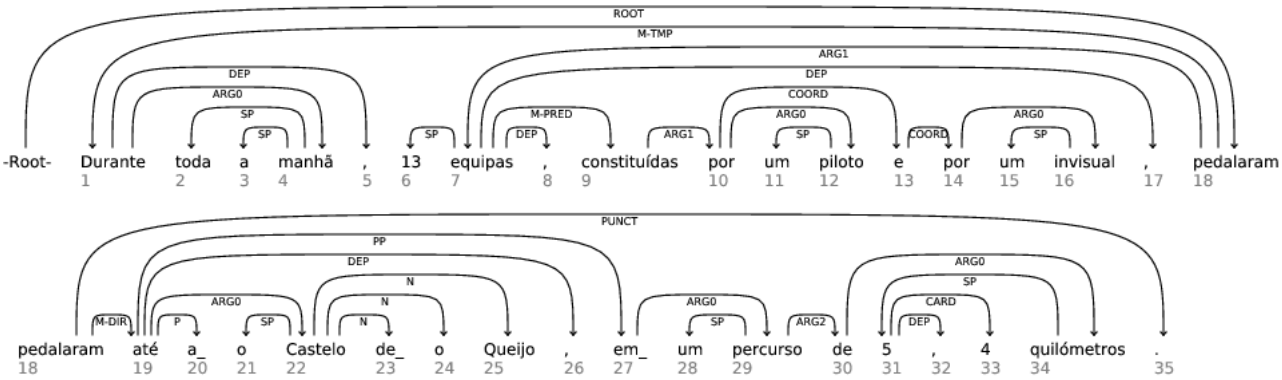
Para a Frase (12):



os triplos são:

(mas,	ARG0,	era)
(era,	ARG2,	Ele)
(Era,	PRD,	escura)
(Era,	S,	mas)
(Root,	ROOT,	Era)

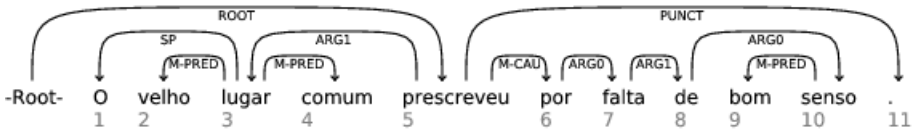
Para a Frase (13):



os triplos são:

(Root,	ROOT,	pedalaram)
(pedalaram,	M-TMP,	Durante)
(pedalaram,	ARG1,	equipas)
(pedalaram,	M-DIR,	até)
(quilómetros,	SP,	5)
(manhã,	SP,	toda)
(Durante,	ARG0,	manhã)
(até,	ARG0,	Castelo_de__o_Queijo)
(percurso,	ARG2,	quilómetros)
(e,	COORD,	piloto)
(e,	COORD,	invisual)
(Castelo_de__o_Queijo,	ARG0,	percurso)
(5,	DEP,	,)
(5,	CARD,	4)
(constituídas,	ARG1,	e)
(equipas,	SP,	13)
(equipas,	M-PRED,	constituídas)

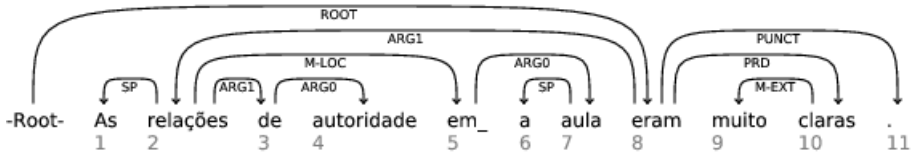
Para a Frase (14):



os triplos são:

(prescreveu,	ARG1,	lugar)
(prescreveu,	M-CAU,	por)
(por,	ARG0,	falta)
(lugar,	M-PRED,	velho)
(lugar,	M-PRED,	comum)
(falta,	ARG1,	senso)
(senso,	M-PRED,	bom)
(Root,	ROOT,	prescreveu)

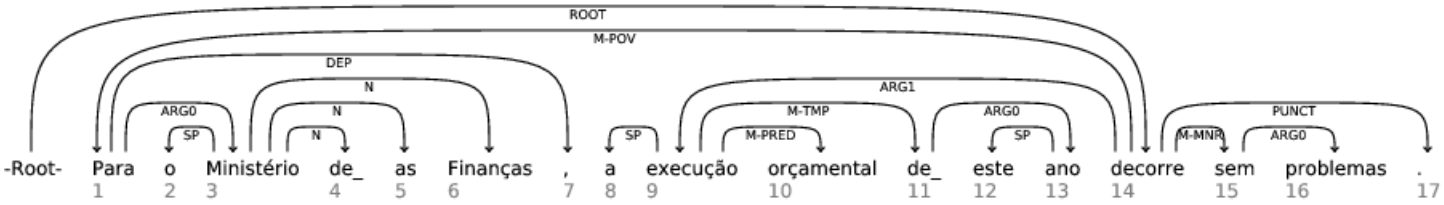
Para a Frase (15):



os triplos são:

(eram,	ARG1,	relações)
(eram,	PRD,	claras)
(em_,	ARG0,	aula)
(claras,	M-EXT,	muito)
(Root,	ROOT,	eram)
(relações,	ARG1,	autoridade)
(relações,	M-LOC,	em_)

Para a Frase (16):



os triplos são:

(de_,	ARG0,	ano)
(ano,	SP,	este)
(execução,	M-PRED,	orçamental)
(execução,	M-TMP,	de_)
(decorre,	M-POV,	Para)
(decorre,	ARG1,	execução)
(decorre,	M-MNR,	sem)
(Root,	ROOT,	decorre)
(Para,	ARG0,	Ministério_de__as_Finanças)
(sem,	ARG0,	problemas)

Apêndice B

Documento RDF/XML

```
<?xml version="1.0"?>

<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:ARG2="Argument_2#"
      xmlns:M-ADV="Modifier_Adverb#"
      xmlns:COORD="Coordination#"
      xmlns:ARG0="Argument_0#"
      xmlns:ROOT="Root_Sentence#"
      xmlns:ARG1="Argument_1#"
      xmlns:M-PRED="Modifier_Predicate#"
      xmlns:M-LOC="Modifier_Localization#"
      xmlns:M-MNR="Modifier_Manner#"
      xmlns:M-NULL="Modifier_Null#"
      xmlns:M-TMP="Modifier_Time#"
      xmlns:ARGA="Argument_A#"
      xmlns:N="Name#"
      xmlns:ARG3="Argument_3#"
      xmlns:PRD="Predicate#"
      xmlns:SP="Specifier#"
      xmlns:M="Modifier#"
      xmlns:M-PNC="Modifier_Purpose#"
      xmlns:S="Sentence#"
      xmlns:M-DIR="Modifier_Direction#"
      xmlns:DEP="Dependency#"
      xmlns:CARD="Cardinal#"
      xmlns:M-CAU="Modifier_Cause#"
      xmlns:M-EXT="Modifier_Extension#"
```

```
      xmlns:M-POV="Modifier_Point_of_View#"
      xmlns:Value="Word_Value#"
>



   <rdf:Description rdf:about="Sentence_0-Word_2"
      Value:Word="Falámos">

      <ARG2:Argument_2 rdf:resource="Sentence_0-Word_9"
         Value:Word="e"/>

   </rdf:Description>

   <rdf:Description rdf:about="Sentence_0-Word_11"
      Value:Word="assuntos">

      <M-ADV:Modifier_Adverb rdf:resource="Sentence_0-Word_12"
         Value:Word="de"/>

   </rdf:Description>

   <rdf:Description rdf:about="Sentence_0-Word_9"
      Value:Word="e">

      <COORD:Coordination rdf:resource="Sentence_0-Word_4"
         Value:Word="caça"/>

      <COORD:Coordination rdf:resource="Sentence_0-Word_8"
         Value:Word="sócios"/>

      <COORD:Coordination rdf:resource="Sentence_0-Word_11"
         Value:Word="assuntos"/>

   </rdf:Description>

   <rdf:Description rdf:about="Sentence_0-Word_12"
      Value:Word="de">
```

```
      <ARG0:Argument_0 rdf:resource="Sentence_0-Word_13"
         Value:Word="homens"/>

   </rdf:Description>

   <rdf:Description rdf:about="Sentence_0" >

      <ROOT:Root_Sentence rdf:resource="Sentence_0-Word_2"
         Value:Word="Falámos"/>

   </rdf:Description>

   <rdf:Description rdf:about="Sentence_1-Word_20"
      Value:Word="pai">

      <ARG1:Argument_1 rdf:resource="Sentence_1-Word_22"
         Value:Word="Acácio"/>

   </rdf:Description>

   <rdf:Description rdf:about="Sentence_1-Word_17"
      Value:Word="sr._Pinheiro">

      <M-PRED:Modifier_Predicate rdf:resource="Sentence_1-Word_20"
         Value:Word="pai"/>

   </rdf:Description>

   <rdf:Description rdf:about="Sentence_1-Word_5"
      Value:Word="circulam">

      <M-LOC:Modifier_Localization rdf:resource="Sentence_1-Word_1"
         Value:Word="Perto"/>

      <ARG1:Argument_1 rdf:resource="Sentence_1-Word_15"
         Value:Word="e"/>

   </rdf:Description>
```

```
<rdf:Description rdf:about="Sentence_1" >

   <ROOT:Root_Sentence rdf:resource="Sentence_1-Word_5"
      Value:Word="circulam"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_1-Word_15"
   Value:Word="e">

   <COORD:Coordination rdf:resource="Sentence_1-Word_6"
      Value:Word="polícias"/>

   <COORD:Coordination rdf:resource="Sentence_1-Word_12"
      Value:Word="irmã"/>

   <COORD:Coordination rdf:resource="Sentence_1-Word_17"
      Value:Word="sr._Pinheiro"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_1-Word_1"
   Value:Word="Perto">

   <ARG2:Argument_2 rdf:resource="Sentence_1-Word_4"
      Value:Word="morte"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_1-Word_12"
   Value:Word="irmã">

   <M-PRED:Modifier_Predicate rdf:resource="Sentence_1-Word_11"
      Value:Word="chorosa"/>

   <ARG1:Argument_1 rdf:resource="Sentence_1-Word_14"
      Value:Word="Cidália"/>

</rdf:Description>
```

```
<rdf:Description rdf:about="Sentence_1-Word_7"
   Value:Word="sem">

   <ARG0:Argument_0 rdf:resource="Sentence_1-Word_8"
      Value:Word="fardas"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_1-Word_6"
   Value:Word="polícias">

   <M-MNR:Modifier_Manner rdf:resource="Sentence_1-Word_7"
      Value:Word="sem"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_2-Word_7"
   Value:Word="desde">

   <ARG0:Argument_0 rdf:resource="Sentence_2-Word_10"
      Value:Word="hora"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_2-Word_10"
   Value:Word="hora">

   <M-NULL:Modifier_Null rdf:resource="Sentence_2-Word_9"
      Value:Word="primeira"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_2-Word_4"
   Value:Word="movimentos">

   <ARG1:Argument_1 rdf:resource="Sentence_2-Word_6"
      Value:Word="Saddam"/>
```

```
</rdf:Description>

<rdf:Description rdf:about="Sentence_2-Word_2"
   Value:Word="acompanhou">

   <ARG1:Argument_1 rdf:resource="Sentence_2-Word_1"
      Value:Word="Washington"/>

   <ARG2:Argument_2 rdf:resource="Sentence_2-Word_4"
      Value:Word="movimentos"/>

   <M-TMP:Modifier_Time rdf:resource="Sentence_2-Word_7"
      Value:Word="desde"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_2" >

   <ROOT:Root_Sentence rdf:resource="Sentence_2-Word_2"
      Value:Word="acompanhou"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_3-Word_4"
   Value:Word="professora">

   <M-LOC:Modifier_Localization rdf:resource="Sentence_3-Word_5"
      Value:Word="em_"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_3-Word_5"
   Value:Word="em_">

   <ARG0:Argument_0 rdf:resource="Sentence_3-Word_7"
      Value:Word="Escola_secundária_de__a_Damaia"/>

</rdf:Description>
```

```
<rdf:Description rdf:about="Sentence_3-Word_13"
   Value:Word="lamenta">

   <ARG1:Argument_1 rdf:resource="Sentence_3-Word_1"
      Value:Word="Paula_Teixeira"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_3-Word_1"
   Value:Word="Paula_Teixeira">

   <M-PRED:Modifier_Predicate rdf:resource="Sentence_3-Word_4"
      Value:Word="professora"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_3" >

   <ROOT:Root_Sentence rdf:resource="Sentence_3-Word_13"
      Value:Word="lamenta"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_4-Word_2"
   Value:Word="e">

   <COORD:Coordination rdf:resource="Sentence_4-Word_1"
      Value:Word="PSD"/>

   <COORD:Coordination rdf:resource="Sentence_4-Word_3"
      Value:Word="PS"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_4-Word_5"
   Value:Word="trabalhos">

   <ARG1:Argument_1 rdf:resource="Sentence_4-Word_8"
      Value:Word="Constituição"/>
```

```
</rdf:Description>

<rdf:Description rdf:about="Sentence_4" >

   <ROOT:Root_Sentence rdf:resource="Sentence_4-Word_4"
      Value:Word="desaceleram"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_4-Word_4"
   Value:Word="desaceleram">

   <ARGA:Argument_A rdf:resource="Sentence_4-Word_2"
      Value:Word="e"/>

   <ARG1:Argument_1 rdf:resource="Sentence_4-Word_5"
      Value:Word="trabalhos"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_5-Word_14"
   Value:Word="e">

   <COORD:Coordination rdf:resource="Sentence_5-Word_8"
      Value:Word="primeiro-ministro"/>

   <COORD:Coordination rdf:resource="Sentence_5-Word_17"
      Value:Word="chefe"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_5" >

   <ROOT:Root_Sentence rdf:resource="Sentence_5-Word_1"
      Value:Word="Foi"/>

</rdf:Description>
```

```
<rdf:Description rdf:about="Sentence_5-Word_17"
   Value:Word="chefe">

   <ARG1:Argument_1 rdf:resource="Sentence_5-Word_20"
      Value:Word="diplomacia"/>

   <N:Name rdf:resource="Sentence_5-Word_22"
      Value:Word="David_Levy"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_5-Word_2"
   Value:Word="acompanhado">

   <ARG3:Argument_3 rdf:resource="Sentence_5-Word_5"
      Value:Word="visita"/>

   <ARG1:Argument_1 rdf:resource="Sentence_5-Word_14"
      Value:Word="e"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_5-Word_1"
   Value:Word="Foi">

   <PRD:Predicate rdf:resource="Sentence_5-Word_2"
      Value:Word="acompanhado"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_5-Word_8"
   Value:Word="primeiro-ministro">

   <M-PRED:Modifier_Predicate rdf:resource="Sentence_5-Word_9"
      Value:Word="israelita"/>

   <N:Name rdf:resource="Sentence_5-Word_11"
      Value:Word="Benjamin_Netanyahu"/>
```

```
</rdf:Description>

<rdf:Description rdf:about="Sentence_6-Word_4"
   Value:Word="fugiam">

   <ARG1:Argument_1 rdf:resource="Sentence_6-Word_2"
      Value:Word="vítimas"/>

   <M-ADV:Modifier_Adverb rdf:resource="Sentence_6-Word_3"
      Value:Word="não"/>

   <ARG2:Argument_2 rdf:resource="Sentence_6-Word_7"
      Value:Word="normalidade"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_6" >

   <ROOT:Root_Sentence rdf:resource="Sentence_6-Word_4"
      Value:Word="fugiam"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_7-Word_5"
   Value:Word="computadores">

   <SP:Specifier rdf:resource="Sentence_7-Word_2"
      Value:Word="todos"/>

   <SP:Specifier rdf:resource="Sentence_7-Word_4"
      Value:Word="quatro"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_7-Word_1"
   Value:Word="Chegaram">

   <ARG1:Argument_1 rdf:resource="Sentence_7-Word_5"
      Value:Word="computadores"/>
```

```
</rdf:Description>

<rdf:Description rdf:about="Sentence_7" >

   <ROOT:Root_Sentence rdf:resource="Sentence_7-Word_1"
      Value:Word="Chegaram"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_8-Word_9"
   Value:Word="pelouro">

   <ARG1:Argument_1 rdf:resource="Sentence_8-Word_13"
      Value:Word="e"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_8-Word_13"
   Value:Word="e">

   <COORD:Coordination rdf:resource="Sentence_8-Word_12"
      Value:Word="obras"/>

   <COORD:Coordination rdf:resource="Sentence_8-Word_14"
      Value:Word="construção"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_8-Word_4"
   Value:Word="entra">

   <ARG1:Argument_1 rdf:resource="Sentence_8-Word_1"
      Value:Word="Manuel_Basto_Saragoça"/>

   <M-ADV:Modifier_Adverb rdf:resource="Sentence_8-Word_5"
      Value:Word="com"/>

   <ARG2:Argument_2 rdf:resource="Sentence_8-Word_9"
```

```
      Value:Word="pelouro"/>

   <M-PRED:Modifier_Predicate rdf:resource="Sentence_8-Word_16"
      Value:Word="substituindo"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_8" >

   <ROOT:Root_Sentence rdf:resource="Sentence_8-Word_4"
      Value:Word="entra"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_8-Word_16"
   Value:Word="substituindo">

   <ARG2:Argument_2 rdf:resource="Sentence_8-Word_17"
      Value:Word="Wladimiro_Ricardo"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_8-Word_5"
   Value:Word="com">

   <ARG0:Argument_0 rdf:resource="Sentence_8-Word_6"
      Value:Word="Campos"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_9-Word_6"
   Value:Word="constituirá">

   <ARG1:Argument_1 rdf:resource="Sentence_9-Word_2"
      Value:Word="experiência"/>

   <ARG2:Argument_2 rdf:resource="Sentence_9-Word_9"
      Value:Word="contribuição"/>
```

```
</rdf:Description>

<rdf:Description rdf:about="Sentence_9-Word_20"
   Value:Word="para">

   <ARG0:Argument_0 rdf:resource="Sentence_9-Word_21"
      Value:Word="cegos"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_9-Word_18"
   Value:Word="de">

   <ARG0:Argument_0 rdf:resource="Sentence_9-Word_19"
      Value:Word="cães-guias"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_9" >

   <ROOT:Root_Sentence rdf:resource="Sentence_9-Word_6"
      Value:Word="constituirá"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_9-Word_16"
   Value:Word="escola">

   <M:Modifier rdf:resource="Sentence_9-Word_15"
      Value:Word="primeira"/>

   <M-PRED:Modifier_Predicate rdf:resource="Sentence_9-Word_17"
      Value:Word="portuguesa"/>

   <M-PNC:Modifier_Purpose rdf:resource="Sentence_9-Word_18"
      Value:Word="de"/>

</rdf:Description>
```

```
<rdf:Description rdf:about="Sentence_9-Word_19"
   Value:Word="cães-guias">

   <M-PNC:Modifier_Purpose rdf:resource="Sentence_9-Word_20"
      Value:Word="para"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_9-Word_2"
   Value:Word="experiência">

   <ARG1:Argument_1 rdf:resource="Sentence_9-Word_4"
      Value:Word="Carlos_Sobral"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_9-Word_9"
   Value:Word="contribuição">

   <M-PRED:Modifier_Predicate rdf:resource="Sentence_9-Word_8"
      Value:Word="importante"/>

   <ARG2:Argument_2 rdf:resource="Sentence_9-Word_12"
      Value:Word="funcionamento"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_9-Word_12"
   Value:Word="funcionamento">

   <M-PRED:Modifier_Predicate rdf:resource="Sentence_9-Word_11"
      Value:Word="bom"/>

   <ARG1:Argument_1 rdf:resource="Sentence_9-Word_16"
      Value:Word="escola"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_10-Word_7"
```

```
      Value:Word="manutenção">

      <ARG2:Argument_2 rdf:resource="Sentence_10-Word_10"
         Value:Word="escola"/>

      <M-TMP:Modifier_Time rdf:resource="Sentence_10-Word_11"
         Value:Word="até"/>

   </rdf:Description>

   <rdf:Description rdf:about="Sentence_10-Word_11"
      Value:Word="até">

      <ARG0:Argument_0 rdf:resource="Sentence_10-Word_12"
         Value:Word="1999"/>

   </rdf:Description>

   <rdf:Description rdf:about="Sentence_10-Word_3"
      Value:Word="fundos">

      <M-PRED:Modifier_Predicate rdf:resource="Sentence_10-Word_4"
         Value:Word="comunitários"/>

   </rdf:Description>

   <rdf:Description rdf:about="Sentence_10-Word_5"
      Value:Word="asseguram">

      <ARG1:Argument_1 rdf:resource="Sentence_10-Word_3"
         Value:Word="fundos"/>

      <ARG2:Argument_2 rdf:resource="Sentence_10-Word_7"
         Value:Word="manutenção"/>

   </rdf:Description>

   <rdf:Description rdf:about="Sentence_10" >
```

```
      <ROOT:Root_Sentence rdf:resource="Sentence_10-Word_5"
         Value:Word="asseguram"/>

   </rdf:Description>

   <rdf:Description rdf:about="Sentence_11-Word_18"
      Value:Word="de_">

      <ARG0:Argument_0 rdf:resource="Sentence_11-Word_20"
         Value:Word="cidade"/>

   </rdf:Description>

   <rdf:Description rdf:about="Sentence_11-Word_15"
      Value:Word="de_">

      <ARG0:Argument_0 rdf:resource="Sentence_11-Word_17"
         Value:Word="festas"/>

   </rdf:Description>

   <rdf:Description rdf:about="Sentence_11-Word_2"
      Value:Word="Posso">

      <ARG1:Argument_1 rdf:resource="Sentence_11-Word_3"
         Value:Word="ver"/>

   </rdf:Description>

   <rdf:Description rdf:about="Sentence_11-Word_14"
      Value:Word="eventos">

      <SP:Specifier rdf:resource="Sentence_11-Word_11"
         Value:Word="todos"/>

      <M-NULL:Modifier_Null rdf:resource="Sentence_11-Word_13"
         Value:Word="outros"/>

      <M-ADV:Modifier_Adverb rdf:resource="Sentence_11-Word_15"
```

```
      Value:Word="de_"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_11" >

   <ROOT:Root_Sentence rdf:resource="Sentence_11-Word_2"
      Value:Word="Posso"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_11-Word_5"
   Value:Word="concerto">

   <ARG1:Argument_1 rdf:resource="Sentence_11-Word_8"
      Value:Word="Deep_Purple"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_11-Word_17"
   Value:Word="festas">

   <M-ADV:Modifier_Adverb rdf:resource="Sentence_11-Word_18"
      Value:Word="de_"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_11-Word_10"
   Value:Word="e">

   <COORD:Coordination rdf:resource="Sentence_11-Word_5"
      Value:Word="concerto"/>

   <COORD:Coordination rdf:resource="Sentence_11-Word_14"
      Value:Word="eventos"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_11-Word_3"
```

```
   Value:Word="ver">

   <ARG2:Argument_2 rdf:resource="Sentence_11-Word_10"
      Value:Word="e"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_12-Word_4"
   Value:Word="mas">

   <ARG0:Argument_0 rdf:resource="Sentence_12-Word_5"
      Value:Word="era"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_12-Word_5"
   Value:Word="era">

   <ARG2:Argument_2 rdf:resource="Sentence_12-Word_6"
      Value:Word="Ele"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_12-Word_1"
   Value:Word="Era">

   <PRD:Predicate rdf:resource="Sentence_12-Word_2"
      Value:Word="escura"/>

   <S:Sentence rdf:resource="Sentence_12-Word_4"
      Value:Word="mas"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_12" >

   <ROOT:Root_Sentence rdf:resource="Sentence_12-Word_1"
      Value:Word="Era"/>
```

```
</rdf:Description>

<rdf:Description rdf:about="Sentence_13" >

   <ROOT:Root_Sentence rdf:resource="Sentence_13-Word_18"
      Value:Word="pedalaram"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_13-Word_18"
   Value:Word="pedalaram">

   <M-TMP:Modifier_Time rdf:resource="Sentence_13-Word_1"
      Value:Word="Durante"/>

   <ARG1:Argument_1 rdf:resource="Sentence_13-Word_7"
      Value:Word="equipas"/>

   <M-DIR:Modifier_Direction rdf:resource="Sentence_13-Word_19"
      Value:Word="até"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_13-Word_34"
   Value:Word="quilómetros">

   <SP:Specifier rdf:resource="Sentence_13-Word_31"
      Value:Word="5"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_13-Word_4"
   Value:Word="manhã">

   <SP:Specifier rdf:resource="Sentence_13-Word_2"
      Value:Word="toda"/>

</rdf:Description>
```

```
<rdf:Description rdf:about="Sentence_13-Word_1"
   Value:Word="Durante">

   <ARG0:Argument_0 rdf:resource="Sentence_13-Word_4"
      Value:Word="manhã"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_13-Word_19"
   Value:Word="até">

   <ARG0:Argument_0 rdf:resource="Sentence_13-Word_22"
      Value:Word="Castelo_de__o_Queijo"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_13-Word_29"
   Value:Word="percurso">

   <ARG2:Argument_2 rdf:resource="Sentence_13-Word_34"
      Value:Word="quilómetros"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_13-Word_13"
   Value:Word="e">

   <COORD:Coordination rdf:resource="Sentence_13-Word_12"
      Value:Word="piloto"/>

   <COORD:Coordination rdf:resource="Sentence_13-Word_16"
      Value:Word="invisual"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_13-Word_22"
   Value:Word="Castelo_de__o_Queijo">

   <ARG0:Argument_0 rdf:resource="Sentence_13-Word_29"
```

```
      Value:Word="percurso"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_13-Word_31"
   Value:Word="5">

   <DEP:Dependency rdf:resource="Sentence_13-Word_32"
      Value:Word=","/>

   <CARD:Cardinal rdf:resource="Sentence_13-Word_33"
      Value:Word="4"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_13-Word_9"
   Value:Word="constituídas">

   <ARG1:Argument_1 rdf:resource="Sentence_13-Word_13"
      Value:Word="e"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_13-Word_7"
   Value:Word="equipas">

   <SP:Specifier rdf:resource="Sentence_13-Word_6"
      Value:Word="13"/>

   <M-PRED:Modifier_Predicate rdf:resource="Sentence_13-Word_9"
      Value:Word="constituídas"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_14-Word_5"
   Value:Word="prescreveu">

   <ARG1:Argument_1 rdf:resource="Sentence_14-Word_3"
      Value:Word="lugar"/>
```

```
      <M-CAU:Modifier_Cause rdf:resource="Sentence_14-Word_6"
         Value:Word="por"/>

   </rdf:Description>

   <rdf:Description rdf:about="Sentence_14-Word_6"
      Value:Word="por">

      <ARG0:Argument_0 rdf:resource="Sentence_14-Word_7"
         Value:Word="falta"/>

   </rdf:Description>

   <rdf:Description rdf:about="Sentence_14-Word_3"
      Value:Word="lugar">

      <M-PRED:Modifier_Predicate rdf:resource="Sentence_14-Word_2"
         Value:Word="velho"/>

      <M-PRED:Modifier_Predicate rdf:resource="Sentence_14-Word_4"
         Value:Word="comum"/>

   </rdf:Description>

   <rdf:Description rdf:about="Sentence_14-Word_7"
      Value:Word="falta">

      <ARG1:Argument_1 rdf:resource="Sentence_14-Word_10"
         Value:Word="senso"/>

   </rdf:Description>

   <rdf:Description rdf:about="Sentence_14-Word_10"
      Value:Word="senso">

      <M-PRED:Modifier_Predicate rdf:resource="Sentence_14-Word_9"
         Value:Word="bom"/>
```

```
</rdf:Description>

<rdf:Description rdf:about="Sentence_14" >

   <ROOT:Root_Sentence rdf:resource="Sentence_14-Word_5"
      Value:Word="prescreveu"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_15-Word_8"
   Value:Word="eram">

   <ARG1:Argument_1 rdf:resource="Sentence_15-Word_2"
      Value:Word="relações"/>

   <PRD:Predicate rdf:resource="Sentence_15-Word_10"
      Value:Word="claras"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_15-Word_5"
   Value:Word="em_">

   <ARG0:Argument_0 rdf:resource="Sentence_15-Word_7"
      Value:Word="aula"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_15-Word_10"
   Value:Word="claras">

   <M-EXT:Modifier_Extension rdf:resource="Sentence_15-Word_9"
      Value:Word="muito"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_15" >

   <ROOT:Root_Sentence rdf:resource="Sentence_15-Word_8"
```

```
      Value:Word="eram"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_15-Word_2"
   Value:Word="relações">

   <ARG1:Argument_1 rdf:resource="Sentence_15-Word_4"
      Value:Word="autoridade"/>

   <M-LOC:Modifier_Localization rdf:resource="Sentence_15-Word_5"
      Value:Word="em_"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_16-Word_11"
   Value:Word="de_">

   <ARG0:Argument_0 rdf:resource="Sentence_16-Word_13"
      Value:Word="ano"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_16-Word_13"
   Value:Word="ano">

   <SP:Specifier rdf:resource="Sentence_16-Word_12"
      Value:Word="este"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_16-Word_9"
   Value:Word="execução">

   <M-PRED:Modifier_Predicate rdf:resource="Sentence_16-Word_10"
      Value:Word="orçamental"/>

   <M-TMP:Modifier_Time rdf:resource="Sentence_16-Word_11"
      Value:Word="de_"/>
```

```
</rdf:Description>

<rdf:Description rdf:about="Sentence_16-Word_14"
   Value:Word="decorre">

   <M-POV:Modifier_Point_of_View rdf:resource="Sentence_16-Word_1"
      Value:Word="Para"/>

   <ARG1:Argument_1 rdf:resource="Sentence_16-Word_9"
      Value:Word="execução"/>

   <M-MNR:Modifier_Manner rdf:resource="Sentence_16-Word_15"
      Value:Word="sem"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_16" >

   <ROOT:Root_Sentence rdf:resource="Sentence_16-Word_14"
      Value:Word="decorre"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_16-Word_1"
   Value:Word="Para">

   <ARG0:Argument_0 rdf:resource="Sentence_16-Word_3"
      Value:Word="Ministério_de__as_Finanças"/>

</rdf:Description>

<rdf:Description rdf:about="Sentence_16-Word_15"
   Value:Word="sem">

   <ARG0:Argument_0 rdf:resource="Sentence_16-Word_16"
      Value:Word="problemas"/>

</rdf:Description>
```

```
</rdf:RDF>
```

Bibliografia

[1] *The Oxford Handbook of Language Typology*, chapter Expanded version of a chapter to appear in: Jae-Jung Song [ed.]. Oxford: Oxford University Press, 2006.

[2] Giuseppe Attardi. Experiments with a multilanguage non-projective dependency parser. In *Proc. of the Tenth Conference on Natural Language Learning, New York, (NY)*, 2006.

[3] Giuseppe Attardi and Massimiliano Ciaramita. Tree revision learning for dependency parsing. In *Proc. of the Human Language Technology Conference 2007*, 2007.

[4] Jon Barwise and Robin Cooper. Generalized quantifiers and natural language. *Linguistics and Philosophy*, 4(2):159–219, 1981.

[5] Michael A. Covington. A fundamental algorithm for dependency parsing. In *Proceedings, 39th Annual ACM Southeast Conference*, 2001.

[6] Inês Duarte. *Língua Portuguesa - Instrumentos de Análise*. Universidade Aberta, 2000.

[7] Maria Simi Atanas Chanev Giuseppe Attardi, Felice Dell'Orletta and Massimiliano Ciaramita. Multilingual dependency parsing and domain adaptation using desr. In *Proceedings the CoNLL Shared Task Session of EMNLP-CoNLL 2007, Prague*, 2007.

[8] Dick Hudson. Dependency grammar, July 2000. A five-session course July 14-18 2000, at ESSLLI2000 in Birmingham.

[9] Johan Hall Joakim Nivre and Jens Nilsson. Maltparser - a data-driven parser-generator for dependency parser. In *In Proceedings of the fifth international conference on Language Resources and Evaluation (LREC2006)*, pages 2216–2219, 2006.

[10] Inês Duarte Isabel Hub Faria Maria Helena, Mira Mateus and Ana Maria Brito. *Gramática da Língua Portuguesa*. Caminho, 2003.

[11] Maria Francisca Xavier Maria Henriqueta Costa Campos. *Sintaxe e Semântica do Português*. Universidade Aberta, 1991.

[12] Daniel Gildea Martha Palmer and Paul Kingsbury. The proposition bank: An annotated corpus of semantic roles. *Comput. Linguist.*, 31(1):71–106, March 2005.

[13] Joakim Nivre. Incrementality in deterministic dependency parsing. Workshop at ACL-2004, July. In Incremental Parsing: Bringing Engineering and Cognition Together.

[14] Joakim Nivre. An efficient algorithm for projective dependency parsing. In *In Proceedings of the 8th International Workshop on Parsing Technologies (IWPT 03)*, pages 149–160, 2003.

[15] Joakim Nivre. Non-projective dependency parsing in expected linear time. In *In Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 351–359, 2009.

[16] Joakim Nivre and Johan Hall. Maltparser: A language-independet system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135, 2007.

[17] Joakim Nivre and Jens Nilsson. Pseudo-projective dependency parsing. In *In Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 99–106, 2005.

[18] Koby Crammer Ryan McDonald and Fernando Pereira. Multilingual dependency analysis with a two-stage discriminative parser. In *Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, 2006.

[19] Koby Crammer Ryan McDonald and Fernando Pereira. Online large-margin training of dependency parsers. In *11th Conference of the European Chapter of the Association for Computational Linguistics: EACL 2006*, pages 81–88, 2006.

[20] Kenji Sagae and Jun'ichi Tsuji. Dependency parsing and domain adaptation with lr models and parser ensembles. In *Proceedings of the CoNLL 2007 Shared Task. Joint Conferences on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL'07)*.

[21] Ivan Titov and James Henderson. Fast and robust multilingual dependency parsing with a generative latent variable model. In *CoNLL 2007 Shared Task. Joint Conf. on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL-07)*, 2007.

[22] Ivan Titov and James Henderson. Fast and robust multilingual dependency parsing with a generative latent variable model. In *In Proc. Joint Conf. on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL 2007)*, 2007.

[23] Ivan Titov and James Henderson. A latent variable model for generative dependency parsing. In *International Conference on Parsing Technologies (IWPT-07)*, 2007.

[24] Kenji Sagae Takuya Matsuzaki Ysuke Miyao, Rune Saetre and Jun'ichi Tsuji. Task-oriented evaluation of syntactic parsers and their representations. In *Proceedings of the 45th Meeting of the Association for Computational Linguistics (ACL'08:HLT)*, 2008.